Linux 典藏大系

ChinaUnix技术社区鼎力推荐

通过大量示例介绍Linux命令、编辑器与Shell编程



Linux 命令、编辑器与Shell编程

王刚 等编著



19.2小时配套多媒体教学视频

11小时Linux指令范例教学视频（赠送）

◎ 内容全面：覆盖Linux命令、运行机制、管理方法、编辑器及Shell编程

◎ 讲解详细：注重分析和实战，详解200余个Linux常见命令的常见选项和用法

◎ 循序渐进：遵循系统管理任务→解决思路→命令介绍→运行机制的学习顺序

◎ 技巧性强：包括大量的经验和技巧，并对容易忽略的细节给出了专门的提示

◎ 注重实战：讲解时穿插了大量的应用示例，Shell编程部分给出了7个案例

清华大学出版社

Linux 命令、编辑器与 Shell 编程

王　刚　等编著

清华大学出版社
北　京

内 容 简 介

本书以实用为原则，以 Linux 系统管理为核心，在深入剖析 Linux 系统运行原理的基础上，重点介绍了 Linux 系统管理中常见命令的各种用法、编辑器的使用及利用 Shell 脚本管理 Linux 系统等内容，还介绍了 Linux 系统管理的常见技巧、解决问题的基本方法和思路，是一本不可多得的 Linux 学习资料。

本书配带 1 张 DVD 光盘，收录了本书重点内容的教学视频和涉及的源代码，光盘中还赠送了大量超值的 Linux 学习视频。

本书共 19 章，分为 3 篇。主要内容有：虚拟化技术、Linux 系统安装与入门、Linux 系统中的人机交互程序、常用命令、正则表达式、查找和筛选命令、权限管理、磁盘和文件系统管理、Linux 系统管理、数据备份、应用程序管理、网络管理、Vim 编辑器、Emacs 编辑器、Eclipse 编辑器和 Linux 系统中常见的编辑器、Shell 脚本编程等。Shell 脚本编程部分引入的实例有：依赖性检查脚本、系统网络监控、文件系统监控、无人值守网络故障分析脚本、自动备份数据脚本、防火墙配置脚本、快速初始化系统脚本等实例。

本书适合 Linux 系统管理人员、维护人员、开发人员和 Linux 爱好者阅读，也适合大中专院校和培训学校作为教材使用。对于经常和 Linux 系统打交道的人员，本书也不失为一本很好的查询手册。

图书在版编目（CIP）数据

Linux 命令、编辑器与 Shell 编程 / 王刚等编著. —北京：清华大学出版社，2012.3
（Linux 典藏大系）
ISBN 978-7-302-27615-9

Ⅰ. ①L…　Ⅱ. ①王…　Ⅲ. ①Linux 操作系统－程序设计　Ⅳ. ①TP316.89

中国版本图书馆 CIP 数据核字（2011）第 271710 号

责任编辑：夏兆彦
责任校对：徐俊伟
责任印制：
出版发行：清华大学出版社
　　网　　址：http://www.tup.com.cn, http://www.wqbook.com
　　地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
　　社 总 机：010-62770175　　**邮　　购：**010-62786544
　　投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn
　　质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　张：**40.75　　**字　数：**1021 千字
　　（附光盘 1 张）
版　　次：2012 年 3 月第 1 版　　**印　次：**2012 年 3 月第 1 次印刷
印　　数：1～ 000
定　　价：　　元

产品编号：044305-01

前　言

为什么要写这本书

近年来随着 Internet 的发展，Linux 系统已经逐渐成长为服务器的主流操作系统之一。众多互联网巨头将其作为关键服务器的操作系统。虽然目前 Linux 系统的市场占有率仍然众说纷纭，但从招聘会的情况来看，Linux 系统正在发展壮大。要想成为一名 Linux 系统管理员，除了熟悉各种服务软件的架设和使用外，还应该具备扎实的系统管理知识，这样才能具备更强的竞争力。在实际管理过程中，也可以更好地应对管理过程中的各种情况。

目前市场上关于 Linux 系统服务方面的图书不少，但真正从系统管理角度出发，介绍常用命令、Shell 脚本编程的图书却很少。本书以实用为原则，从系统管理出发，深入剖析 Linux 系统的运行原理，介绍了 Linux 系统中的常用命令、管理命令的使用方法、利用 Shell 脚本管理系统等内容，让读者可以深入、全面地了解 Linux 系统管理的主流技术和方法，为提高 Linux 系统水平打下坚实的基础。

本书有何特色

1．提供教学视频和本书源代码，提高学习效率

为了便于读者学习本书中的内容，提高学习效率，作者对书中的重点内容特意制作了教学视频收录于配书 DVD 光盘中，同时也将书中涉及的源代码收录于本书的配套光盘中。另外，光盘中还赠送了大量超值的 Linux 学习视频。

2．介绍命令的同时，也介绍了命令的常见用法

本书在介绍 Linux 系统命令时，还一并介绍了命令在不同情形下的多种常见用法。通过学习命令的常见用法，读者不必再为记忆众多的命令、选项而烦恼。

3．介绍Linux系统的管理时，注重应用性

介绍 Linux 系统管理时，注重讲解管理技巧、方法。这些技巧、方法都是系统管理员们经常用到的，读者可以从中学习到更多前人的经验。

4．讲解系统管理时，注重介绍系统的运行原理

为了帮助读者从书中的例子中学会举一反三，介绍系统管理知识时，本书更注重介绍 Linux 系统的运行原理。读者了解系统的运行原理后，就能够在实际管理过程中按需定制。

5．从实际管理入手，介绍多个具有较高应用价值的实例

在本书的最后一篇中，介绍 Shell 编程的同时，引入了许多实际应用的例子。这些例子具有很高的实用性和参考性，读者可以修改这些例子作练习，也可以稍加修改应用到实际环境中。

6．提供完善的技术支持和售后服务

本书提供了专门的技术支持邮箱：fans_linux@163.com。读者在阅读本书过程中有任何疑问都可以通过该邮箱获得帮助。

本书内容及知识体系

第1篇　Linux命令（第1～10章）

本篇介绍了 Linux 系统中的常见命令。主要包括 Linux 系统安装、入门、常见命令、正则表达式、权限管理、磁盘和文件系统管理、系统管理、数据备份等。

第2篇　文本编辑器（第11～14章）

本篇介绍了 Linux 系统中最常见的文本编辑器的使用。主要包括 Vi 和 Vim 编辑器、Emacs 编辑器、Eclipse 编辑器、Nano 编辑器、图形界面中的常见编辑器等。

第3篇　Shell编程（第15～19章）

本篇主要介绍了 Linux 系统中的 Shell 编程。主要包括 Shell 编程基础、Shell 中的变量、Linux 系统脚本和用户环境、函数和脚本参数、控制 Shell 脚本的执行顺序、调试 Shell 脚本、Shell 脚本的应用实例等。在介绍 Linux 系统脚本时，着重分析了 Linux 系统启动过程中使用的脚本、系统初始化的详细过程。在第 19 章中还引入了 4 个编程实例，介绍 Shell 编程在 Linux 系统管理中的应用。

配书光盘内容介绍

为了方便读者阅读、学习本书，本书附带 1 张 DVD 光盘。内容如下：

- 本书中所有脚本的源代码；
- 本书每章内容的教学视频；
- 免费赠送的大量超值 Linux 学习视频。

适合阅读本书的读者

- 经常接触 Linux 系统的读者；
- Linux 系统管理员；
- Linux 系统开发人员；

- 运营维护管理人员；
- Linux 系统爱好者；
- 大中专院校的学生；
- 培训机构的学员；
- 需要一本查询手册的人员。

阅读本书的建议

- 学习本书前，需要了解一些计算机基础知识，例如计算机硬件知识、磁盘结构和数据存储、计算机网络、对操作系统有一定的认识。
- 如果读者在之前没有接触过 Linux 系统，建议从第 1 章开始，学习、练习命令的每个用法。
- 本书注重引导读者自主思考，阅读时需要读者自主思考，理解书中解决问题的思路和方法。
- 对于书中的脚本代码，应该先了解其功能，然后再参考注释仔细分析一次。这有助于理解脚本编写思路，学习也会更加深刻。

本书作者

本书由王刚主笔编写。其他参与编写和资料整理的人员有邱伟、薛富实、李静、徐阳、陈一东、王静、刘颖、胡国庆、周萌、李季萌、谢建、杜礼、王健、黄进、王以荣、郑波、赖俊文、张光泽、刘智慧、杨雪梅、祝翠、张晓静、陈虹翔、陈国建、唐清荣、刘畅、颜盟盟、姚志娟、张昆。

在本书编写期间得到了我的朋友和同事们的大力帮助，在此表示感谢！

虽然我们对本书中所述内容都尽量核实，并多次进行文字校对，但仍可能还存在疏漏和不足之处，恳请读者批评指正。

编著者

目　　录

第 1 篇　Linux 命令

第 2 篇 文本编辑器

第 3 篇　Shell 编程

第 1 篇 Linux 命令

第 1 章　Linux 简介与安装

1991 年荷兰赫尔辛基大学一名大二学生林纳斯·托瓦兹（Linus B. Torvalds）编写了 Linux 系统，并将其放在互联网上。经过这 20 年的发展，Linux 系统已经扩展到各个领域，从拥有成千上万个 CPU 的超级计算机到只有几个芯片组成的单片机，随处可见 Linux 系统的身影。正是因为 Linux 的流行，笔者特意编写了此书，让读者可以更好地学习 Linux。

本章主要涉及的知识点如下。

- ❑ 介绍 Linux 系统、主要发行版及 Linux 系统的优势等内容。
- ❑ 简述 Linux 系统中的存储及目录结构。
- ❑ 介绍 VMware Workstation 软件和虚拟化在企业中的应用。
- ❑ 以 VMware 虚拟机安装 Linux 操作系统为例，讲解 Linux 系统的安装过程及安装过程中的建议等内容。

1.1　Linux 系统简介

通常所说的 Linux 操作系统，是对使用 Linux 内核的一类操作系统的统称，这些操作系统的主要结构包括：Linux 内核、人机交互程序、应用程序等。本节将简单介绍 Linux 的用途和优势。

说明： Linux 内核是操作系统的核心部分，主要负责管理进程、存储设备和网络接口等。无论何种操作系统，内核都至关重要，其决定了操作系统的许多性能指标。

小知识： Linux 隶属于 GNU（GNU's Not UNIX）计划，该计划的目标是建立一个自由的操作系统，即自由地使用、复制、修改、发布操作系统及其中的软件。GNU 计划还包括许多软件，例如文本编辑器 GNU Emacs、GCC 等。GNU/Linux 操作系统通常被简称为 Linux，如果没有特殊说明，本书中的 Linux 即指 GNU/Linux。

1.1.1　Linux 能做什么

Linux 究竟能为我们做些什么呢？这是每个用户都关心的问题。目前 Linux 系统的应用主要分为桌面环境和企业环境两个方面，本小节将从这两个方面简单介绍 Linux 系统的应用。

1. 桌面环境

对于家庭用户而言，Linux 提供了比较方便的 KDE 和 GNOME 桌面环境。桌面环境中自带的软件能够满足用户的不同需求。

- 利用 Mozilla Firefox 等浏览器可以轻松浏览互联网上的网页。
- 使用 KMail 可以收发电子邮件。
- 利用 XMMS、Kxine 等多媒体播放器，可以播放音乐和视频等多媒体。
- 开源软件 OpenOffice 是一个功能强大的办公软件，不仅可以对文字进行排版、编辑，还可以用来编辑网页、数据库等。

除了以上这些软件之外，Linux 系统中还有许多有特色的软件，例如图形处理软件 GIMP 等，使用这些软件可以轻松地完成桌面环境中的应用。

2. 企业环境

Linux 作为一个类 UNIX 操作系统，其继承了 UNIX 的许多特性，UNIX 系统中的许多优秀服务器软件都可以在 Linux 系统中运行。除此之外，Linux 系统中还有很多开源服务器软件，企业可以使用这些软件构建自己的网络服务器。下面简单介绍这些服务器软件。

- 利用 BIND 可以构建企业级域名服务器。
- 使用 MySQL 和 PostgreSQL 可以构建企业级数据中心。
- Linux 系统中拥有知名的 Web 服务器软件 Apache，高性能 Web 服务器软件 Nginx 等，使用这些软件可以构建企业级 Web 服务器。
- 使用 Qmail、Postfix 和 Sendmail 可以构建企业级邮件服务器，而另一些使用 Linux 系统构建的专用小型邮件系统（如 EMOS），甚至可以轻松应付百万用户级的邮件解决方案。

除了以上这些软件外，Linux 系统中还有很多高性能服务器软件，例如代理服务器软件 Squid、文件服务器软件 Samba、FTP 服务器软件 vsFTPd 等。这些软件形成了一整套企业服务解决方案。

1.1.2　Linux 的主要发行版

用户是无法使用只有一个 Linux 内核的操作系统的，将 Linux 内核、人机交互程序及各种应用程序组合在一起，就组成了用户可以使用的操作系统，通常将其称为 Linux 发行版。Linux 从诞生至今已有 20 年的历史，使用 Linux 内核的发行版有数百种之多。本小节将简单介绍一些常见的 Linux 发行版。

1. Red Hat Linux

Red Hat 公司成立于 1995 年，是目前规模最大的 Linux 发行版企业。Red Hat 到目前为止发行过两个版本：一个是 Red Hat Linux 桌面版，最后一个版本是 Red Hat Linux 9.0，目前这个版本已经停止开发；另一个是 Red Hat Linux 企业版，到本书编写时 Red Hat Linux 企业版已经发行至第 6 版。

2．Suse Linux

Suse Linux 源于 Slackware Linux，2004 年 Novell 完成对 Suse Linux 的收购。作为一个德国的老牌 Linux 发行版，Suse Linux 拥有不少的用户。Suse 采用了 Red Hat 的不少特性，其自带的包管理工具 Yast2，除了能够更新软件包以外，还能配置防火墙、管理用户，使用起来十分方便。

3．Debian Linux

Debian Linux 是一个自由社区维护的发行版，所有的开发工作都是由世界各地的志愿者完成的。Debian 有一个十分庞大的家族，当前十分流行的 Ubuntu 正是其发行版之一。

4．Red Flag Linux

红旗 Linux 是一个完全国产化的 Linux 发行版，由中科红旗软件技术有限公司负责开发和维护，也是目前国内最有影响力、较为成熟的发行版。红旗 Linux 分为桌面和服务器两个版本，汉化的 Linux 桌面和仿真的 Windows 环境为其在国内赢得了不少用户。

对于国内桌面用户而言，最近雨林木风推出了在 Ubuntu 基础上构建的 Ylmf OS 3.0，其精仿的 Windows XP 界面、添加了许多适合国人使用的软件等，应该算是一大亮点。

小知识： 在选择 Linux 发行版时，应该考虑三个方面：第一是使用人数，使用人数较多的版本通常都可以从互联网上得到较多的帮助；第二是较为稳定、成熟的版本，这样可以尽量避免在使用过程中出现问题；第三，尽量不要使用新版本，这是因为新版本中可能会存在一些 Bug，从而影响系统的稳定性。

1.1.3　Linux 系统的优势

Linux 系统在各领域的广泛使用与其自身的优势分不开，本节将简单介绍这些优势。

1．广泛的硬件支持和强大的计算性能

Linux 是目前所有操作系统中，硬件支持最广的系统之一，能够轻松支持各种硬件平台，这主要源自使用 C 语言编写的内核。在我们的日常生活中，使用 Linux 的设备随处可见，例如电视机顶盒、智能电话、网络摄像头、小型路由器等。

世界前 500 强超级计算机中，大多数使用了 Linux 作为其操作系统，这不得不说明 Linux 拥有强大的计算性能。事实上在许多平台的相关测试中，Linux 的性能也总是脱颖而出。

2．真正的多用户多任务系统

Linux 同 UNIX 一样，是一个真正的多用户多任务操作系统。用户从不同的终端登录，系统会为登录的每个用户分配资源。为保证每个用户间的资源互不影响，Linux 系统执行严格的权限管理，同时 Linux 系统还会平等地为每个用户分配计算资源，所以系统能够互不影响地执行多个用户的多个任务。Linux 系统的这个特性在计算资源丰富的超级计算机和集群上非常有用。

3. 可靠的安全性和良好的稳定性

众所周知的是，Linux 系统对病毒具有先天免疫力，而事实上自 Linux 诞生以来，很少有 Linux 病毒出现。这与 Linux 系统默认的设置和自带的安全工具是分不开的，例如默认权限值、系统防火墙、SELinux 等。

由于来自全世界的程序员、系统架构等方面的专家，一直在对 Linux 系统进行优化和微调，所以 Linux 系统具有很好的稳定性，即使常年不关机也不会宕机。

Linux 系统还有许多优点，例如费用低、可按需求定制等，此处不一一列举，感兴趣的读者可以阅读相关文档了解。

1.2　Linux 的存储设备和目录结构

与国内大多数人使用的 Windows 操作系统不同，Linux 使用另一种方法标识主机上的设备及系统中的目录结构。在安装 Linux 系统之前，应该对基本的存储设备和系统目录结构有一定了解。本节将简单介绍 Linux 系统中的存储设备和目录结构。

1.2.1　Linux 系统中的存储设备

讲到存储设备，很多读者都会想到硬盘，硬盘是当前计算机中存储数据的主要设备。本小节将介绍常见的存储设备在 Linux 系统中的标识方法。

1. 硬盘分类

按读写功能对存储设备进行分类，可以分如下几类。

- 随机存储器，这类存储器的代表是内存。
- 只读存储器，主要指主机上的光驱。
- 可读写储器，指的是硬盘、软驱和 U 盘等。

提示：现在许多系统中的光驱也支持写入（刻录功能）功能，但是需要光盘支持。

目前市场上的存储设备按使用接口类型又可以分为如下几类：

- IDE 设备，一种较为老式的个人计算机硬盘接口，目前仍有少量使用。IDE 接口使用并行方式传输数据，所以有时也称其为“并口”。
- SATA 设备，目前主流的个人计算机硬盘接口。SATA 接口使用串行方式传输数据，人们通常将其形象地称为“串口”。
- SCSI 设备，小型计算机和服务器上的硬盘大多使用这种接口。
- 串行总线设备，主要是指 U 盘、移动硬盘、移动光驱等使用 USB 接口的移动存储设备。

除以上列举的接口类型外，还存在一些比较少见的硬盘接口类型，读者可以自行阅读相关说明了解这些硬盘。

🔔提示：SCSI 接口一般都附带有专用的 SCSI 控制器，以避免存储设备在读写时占用系统的 CPU 资源。使用这类接口的设备有很多，除了硬盘和光驱之外，还包括一些磁带设备和打印机等，本书中仅讨论 SCSI 的硬盘设备。

2．存储设备的标识方法

在 Linux 系统中，主要以接口类型区分存储设备，按存储设备占用的系统接口编号为其分配标识符。具体规则如下。

- IDE 接口设备：使用 IDE 接口的存储设备一般是硬盘和光驱，在 Linux 系统中对这类设备使用的标识符为 hd。按设备使用的 IDE 接口编号不同，分别命名为：hda、hdb、hdc…
- SATA 接口设备：使用这类接口的设备有硬盘、光驱等，这类设备在 Linux 系统中使用的标识符为 sd。按接口编号不同分别命名为 sda、sdb、sdc…
- SCSI 接口设备：Linux 系统使用标识符 sd 标识这类硬盘设备，仍然按使用的接口编号不同，以 sda、sdb、sdc…为其命名。

🔔注意：Linux 系统支持热插拔 SCSI 硬盘，添加硬盘可以不关机就进行操作。

- 串行总线设备：这类设备主要是 U 盘、移动硬盘等，这些设备在 Linux 系统中的标识方法与 SATA 和 SCSI 设备相同。

🔔提示：目前使用的硬盘设备除了 IDE、SATA、SCSI 硬盘之外，还包括一个较新的接口 SAS，其标识方法与 SATA、SCSI 硬盘相同。

3．分区的标识方法

在使用硬盘存储数据时，还需要对硬盘进行分区，按分区的使用方法可以将硬盘分区分为 3 种：主分区、扩展分区和逻辑分区。这 3 种分区的区别及标识方法如下。

- 主分区：硬盘分区中最基本的分区类型，主分区可直接挂载并存储数据，一个硬盘上最多只能有 4 个主分区。Linux 系统中对这 4 个主分区使用标识符 1、2、3、4。例如硬盘 sda 上的第 1 个主分区标识为 sda1，其后 3 个主分区分别标识为 sda2、sda3 和 sda4，其他类型的硬盘依此类推。
- 扩展分区：扩展分区是一种特殊的主分区，如果要使用扩展分区存储数据，必须先将扩展分区划分为逻辑分区（即逻辑分区建立在扩展分区的基础之上）。如果要在一个硬盘上创建 4 个以上的分区，就必须使用扩展分区。由于扩展分区也是一种主分区，因此扩展分区也占用一个主分区号。
- 逻辑分区：在扩展分区的基础之上，可以创建多个逻辑分区，逻辑分区可以直接挂载并存储数据。逻辑分区的标识符从数字 5 开始，例如 sda5、sda6、sda7…其他类型的硬盘以此类推。

Linux 系统中的硬盘分区使用以上标识符命名并保存在目录/dev 中，要使用分区存储数据时，需要将对应的块设备文件挂载到一个目录下。挂载块设备的过程可以描述成：为用户使用该分区存放和读取数据提供一个接口或途径。

1.2.2　Linux 系统中的目录结构

与 Windows 不同，Linux 系统使用一个目录结构来代表整个文件系统，Linux 系统的目录结构通常由一个或多个文件系统组成。本小节将简单介绍 Linux 系统中的目录结构。

提示：Linux 系统中将一个分区称为一个文件系统。

Linux 系统使用斜杠“/”表示整个目录的起点（/称为根目录），通常人们将根目录所在的分区称为根分区。根目录下有许多用于存放各类文件的子目录，其目录结构如图 1.1 所示。

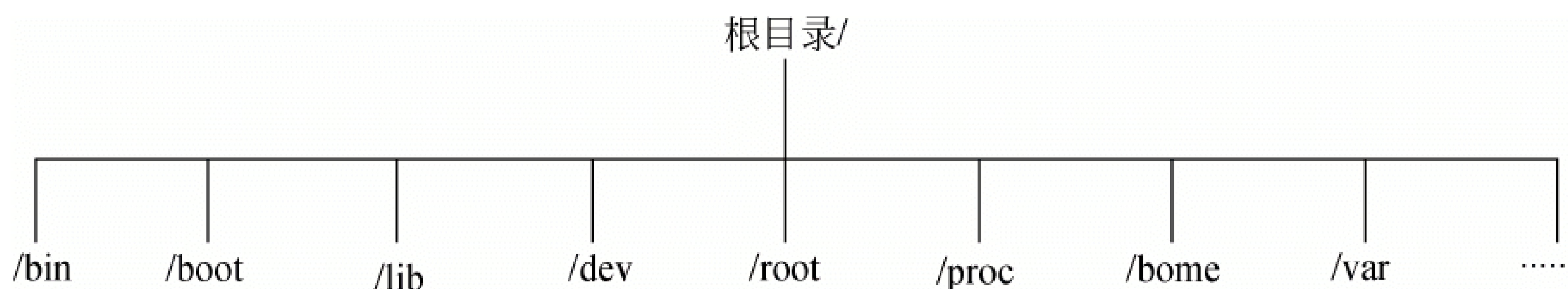


图 1.1　Linux 系统的目录结构

提示：与 Windows 系统不同，Linux 系统中的分区可以挂载到系统中的任意目录，具体目录对应的分区按挂载点不同而不同。

下面将简单介绍根目录中的子目录及其功能。

- bin 目录：主要用于存放普通用户可以使用的命令，例如，rm、sort 等。
- boot 目录：用于存放 Linux 启动所需的文件，包括内核、引导文件等。

提示：通常为 boot 目录单独划分一个分区，称为引导分区。

- dev 目录：通常存放一些设备文件，例如磁盘、光盘、终端、键盘和软驱文件等。
- etc 目录：系统和服务的配置文件都存放在这个目录中（例如系统用户密码、网络接口、防火墙和系统服务的配置文件等），该目录设置有严格的权限，普通用户通常仅能只读。

注意：由于 etc 目录中保存了许多配置文件，因此如果要修改其中的文件，应该遵循先备份后修改的原则。

- home 目录：该目录用于存放普通用户的家目录，通常建议将该目录单独划分为一个分区。

小知识：家目录是用户登录系统后的起始目录，该目录用于存放用户的个人文件、系统和应用程序的初始化文件等。

- lib 目录：通常存放系统及相关软件依赖的库文件。
- lost+found 目录：如果系统出现意外或掉电关机，文件系统可能会损失一些文件或产生一些文件碎片等。当系统重新启动时会修复这些文件，并将这些文件放入此目录以便管理员恢复。多数情况下系统会自动修复这些碎片文件，该目录只存在于分区挂载目录中。

- media 目录：用来挂载一些可移动媒体，例如光驱、U 盘等。
- mnt 目录：通常用来临时挂载一些磁盘设备，也经常用于挂载一些移动存储设备。
- opt 目录：有些系统通常会将额外的软件装在这个目录中，也可以将需要编译安装的软件放在此目录中。
- proc 目录：系统运行时将相关的暂存信息放在此目录中，包括网络、磁盘和进程等信息（该目录由系统产生，并非真实存在于文件系统上）。
- root 目录：这是 root 用户的家目录，通常存放 root 的初始化文件、个人文件等。
- sbin 目录：存放一些系统管理命令，例如 route、mkfs.ext3 等，一般情况下普通用户不能执行这些命令。
- tmp 目录：此目录存放系统或进程在运行时产生的临时文件，用户也可以在这个目录中存放自己的临时文件。

注意：tmp 目录保存的临时文件对所有用户可见，因此此目录中的文件可能会泄露并造成一些安全问题。

- usr 目录：存放一些帮助和文档，有时也存放一些配置文件和程序等。
- var 目录：存放许多类型的文件，例如日志文件、数据库文件和 Web 服务器程序文件等。

上面仅介绍了一些比较常见的目录，在不同的发行版中，根目录中存在的子目录也可能会有差异，可以阅读相关系统的说明文件了解这些目录的作用。

提示：Linux 系统使用不同的目录保存不同的文件，这是许多操作系统推荐的行为，因此读者应该了解这些目录的作用，以便于查找和存放不同的文件。

注意：在没有确认修改文件内容安全之前，不要随意修改目录 proc 中的任何文件，否则可能会造成系统宕机等危害。

1.3　虚拟化技术的应用

通常建议初学者先在虚拟机中学习 Linux 基础知识，待有一定基础知识后再进入实际环境学习。虚拟机是最近几年兴起的新技术，也是虚拟化技术最重要的一部分，虚拟机的实质就是在计算机上使用软件虚拟出一个或多个新的计算机。虚拟出的计算机与真实的计算机一样，不仅拥有独立的 CPU、硬盘等设备，同时还提供多样化的网络连接，如图 1.2 所示。

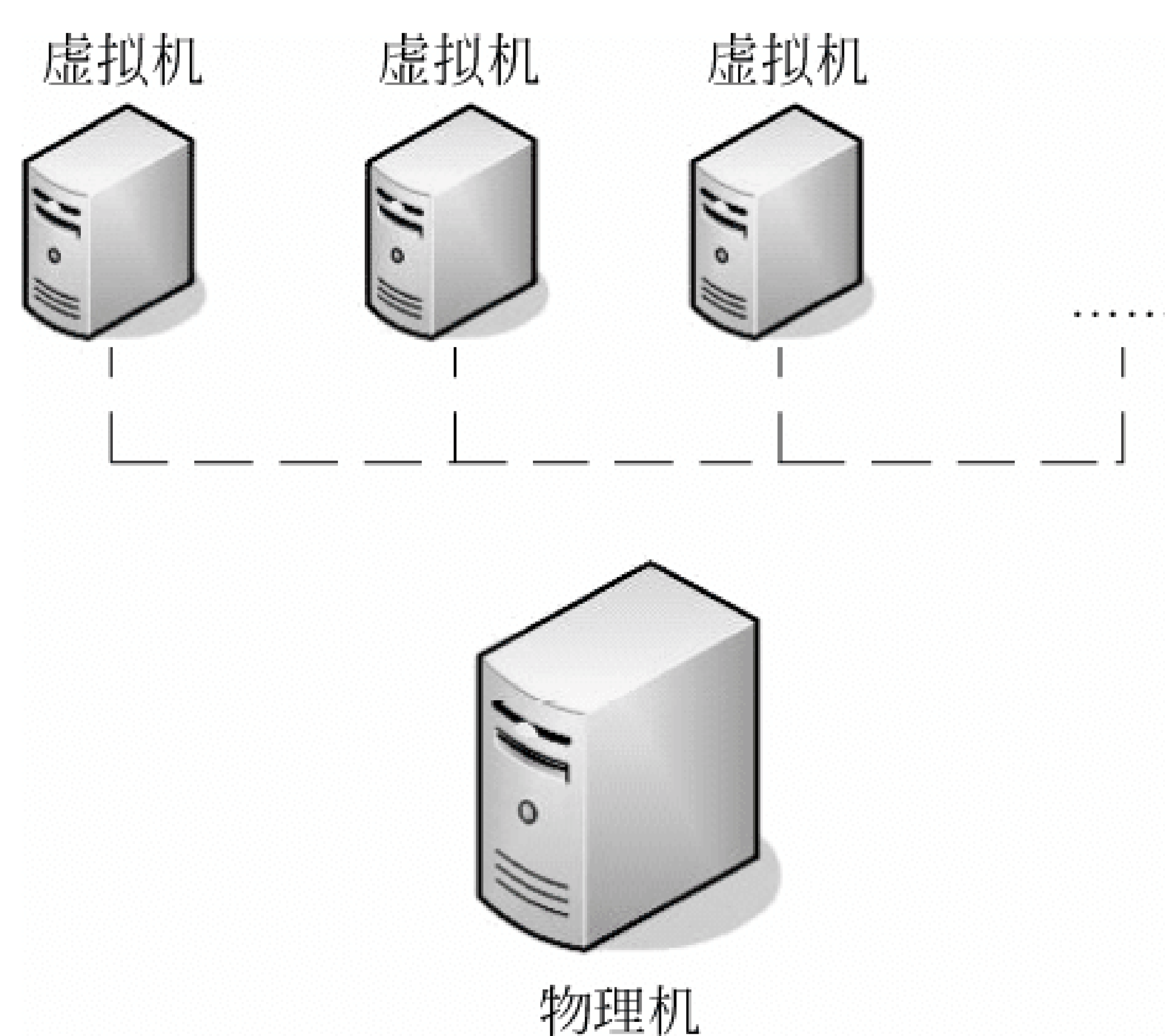


图 1.2　虚拟机示意图

提示：通常将虚拟出的计算机称为虚拟机，运行虚拟机软件的计算机称为物理机。

初学者可以像操作真实计算机那样操作虚拟

机，即使在虚拟机中操作出现失误也不会损坏真实计算机，这是推荐初学者使用虚拟机学习的主要原因之一。

虚拟机不仅能帮助初学者学习，在企业中还具有广泛的应用。本节将简单介绍虚拟化的含义、在企业中的实际应用和虚拟化工具等内容。

提示：虚拟出的多台计算机在运行时不会互相影响，因此可以在虚拟机中安装不同的操作系统。

1.3.1　虚拟化和 VMware 公司

近年来虚拟化技术大行其道，各大虚拟化厂商之间竞争日益激烈。虽然如此，许多初学者仍然不明白虚拟化究竟是什么。本小节将简单介绍虚拟化在企业中的实际应用，以及 VMware 公司及产品介绍等内容。

1. 虚拟化技术的应用

为宣传企业形象、产品理念和自动化办公等，许多企业急需建立自己的网络中心提供多样化服务。这些服务器可能是 Web 服务器、文件服务器、自动办公服务器等。为此企业需要购买多台服务器，并且还需要建设网络中心机房、购买制冷设备等配套设施。购买许多新服务器、配套设施可能会造成资源浪费，为此我们建议以下情况可以使用虚拟化完成。

（1）在一些小型企业中，由于公司业务不大、企业员工较少，使用这些服务的用户数量往往较少，购买多台服务器可能都不能被合理使用，其结果可能是浪费了大笔资金。此时可以考虑购买少许性能较强的服务器，然后配合使用虚拟化技术。

（2）虽然已经有很多处理性能强大的服务器，现在需要一台处理性能一般的服务器安装另一些服务，例如利用 SNMP 协议监控所有服务器的运行状态等。

以上情况均可以使用虚拟化技术，将多个服务安装到虚拟机中。使用虚拟化技术不仅可以节省大笔用于购买和改造各种硬件设施的资金，还可以减少用电、人力资源等后期资金投入，这是其近些年来虚拟化技术流行的重要原因之一。

注意：如果需要将一些关键业务（例如数据库）安装到虚拟机中，应该使用多台运行在不同物理服务器内的虚拟服务器做冗余设置。否则一旦物理服务器宕机，关键业务就会面临中断的危险。除此之外，虚拟化环境下制定的备份策略应该更加灵活，以保证数据的安全。

2. VMware公司的虚拟化产品

VMware 是虚拟化软件的全球领导厂商，许多企业都使用它的产品。VMware 产品按不同需求主要可以分为如下 3 类。

- ❑ VMware Server：一个可以免费获得的虚拟化产品，需要安装在已有的操作系统上。
- ❑ VMware ESX Server：专为企业打造的虚拟化产品，其自身就是一个操作系统，并且只能安装在性能较高的服务器上，配合网络存储还可以实现故障动态迁移等功能。

- ❑ VMware Workstation：针对桌面市场推出的产品，需要安装在已有的操作系统上。

除了这些产品以外，VMware 还推出了许多产品和套件，例如 VMware Player 等，有兴趣的读者可以查阅相关产品手册。

1.3.2　VMware Workstation 简介

VMware Workstation 是 VMware 公司针对桌面市场推出的一款产品，广泛运行于 Windows、Linux 等操作系统中。由于大多数个人用户都使用 VMware Workstation，因此本书重点介绍此版本，对于其他版本，读者可以自行阅读相关说明学习和使用方法。

使用 VMware Workstation 创建的虚拟机可以运行 Linux、UNIX 和 Windows 等多种操作系统，本书编写时其最新版为第 7 版，主界面如图 1.3 所示。

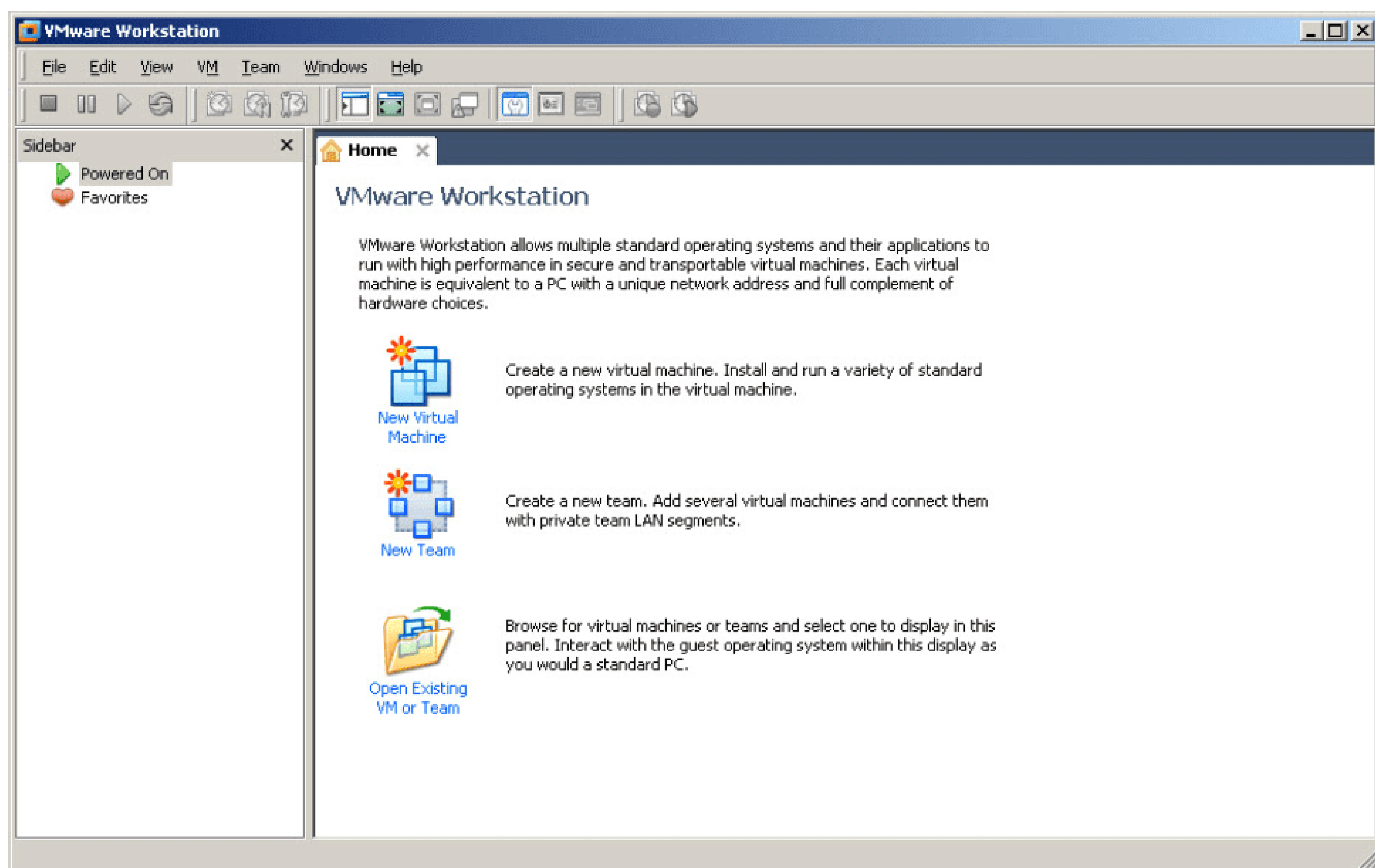


图 1.3　VMware Workstation 运行界面

VMware Workstation 是一个被广泛用于学习的虚拟机软件，功能十分强大。包含的功能如下。

- ❑ 丰富的网络功能：VMware Workstation 不仅提供桥接、NAT 网络功能，还提供了虚拟网络的功能。虚拟网络允许处于不同物理主机的虚拟机互相连接，十分方便模拟大型网络。
- ❑ 快照功能：对虚拟机设置快照后，可以快速将虚拟机恢复到过去的某一时刻。
- ❑ 拖曳文件功能：允许用户在物理机和虚拟机之间拖动文件。

除以上列举的功能之外，VMware Workstation 还提供许多实用的功能，例如录制回放、PXE 等功能，读者可以阅读相关文档了解和使用这些功能。

🔔注意：VMware Workstation 创建的虚拟机虽然与物理主机一样，但有些功能和接口不能使用（例如 3D 图形加速等）。

1.3.3　VMware Workstation 的网络连接方式

VMware Workstation 为满足不同用户的需求，提供了丰富的网络连接方式，使用 VMware Workstation 之前应该对其有所了解。本小节将介绍 VMware Workstation 的网络连接方式。

提示：阅读本小节前应该先掌握计算机网络相关知识（特别是网络地址转换方面的知识）。

小知识：安装完 VMware Workstation 后会自动生成两个网络连接：VMware Network Adapter VMnet8 和 VMware Network Adapter VMnet1（通常称为 VMnet8 和 VMnet1）。虚拟机的网络连接都与这两个连接紧密相关，因此必须保持这两个连接可用，并且不要随意修改这两个连接。

1．NAT方式

使用 NAT（Network Address Translation，网络地址转换）方式时，虚拟机将使用 NAT 方式与主机网络相连，此时虚拟机网络将作为主机的私有网络，如图 1.4 所示。

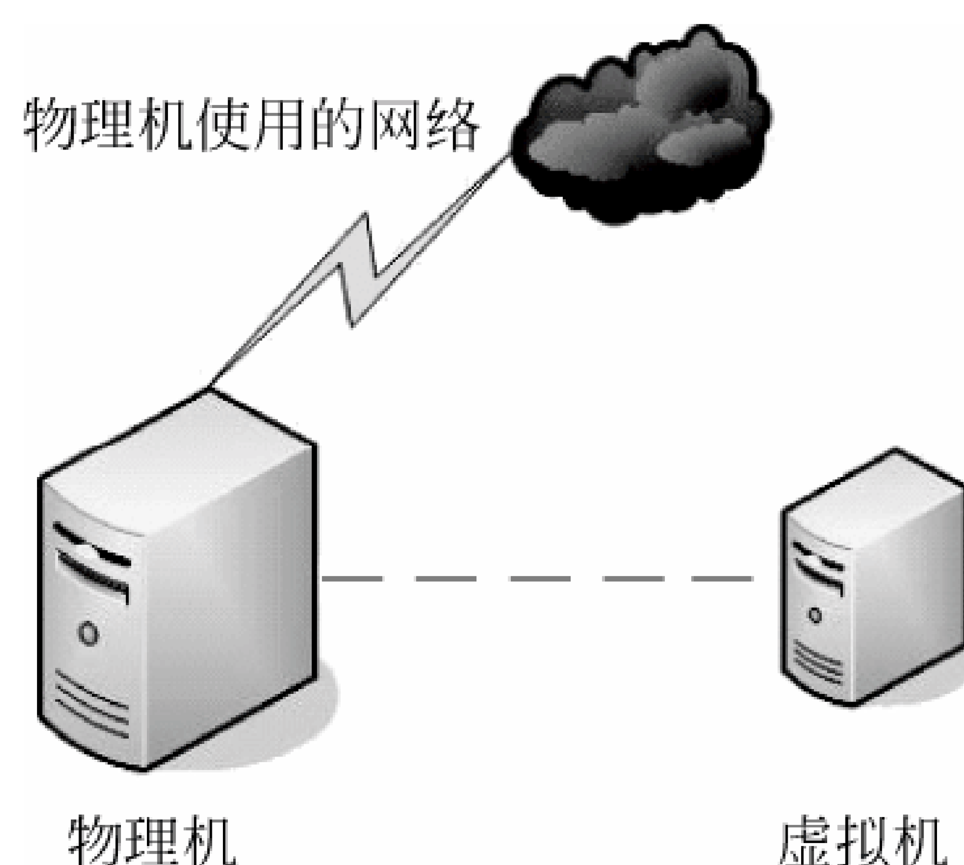


图 1.4　使用 NAT 方式连接网络

从图 1.4 中可以看出，使用 NAT 方式连接到网络时，物理主机相当于一个路由器，物理主机使用的网络连接为外部网络，虚拟机使用的网络为内部网络（私有网络）。

如果以 NAT 方式连接网络，虚拟机将会自动获取 IP 地址、网关、DNS 地址等信息。虽然以 NAT 方式连接网络最简单，但其最大的缺点是外部网络的主机无法访问到虚拟机，因此 NAT 方式连接不适用于为外部提供服务的虚拟机。

小知识：为 NAT 方式连接网络提供支持的是物理主机的两个服务：VMware NAT Service（提供 NAT 服务支持）和 VMware DHCP Service（为虚拟机分配 IP 地址），以及网络连接 VMnet8。因此使用此方式连接网络之前，应该保证上述服务和网络连接可用。

注意：使用此方式时，VMware 通常使用的是基于端口的网络地址转换方式（PNAT）。

2．桥接方式

桥接是虚拟机最常用的网络连接方式，此方式相当于将虚拟机直接连接到与物理主机相连的交换机上，如图 1.5 所示。

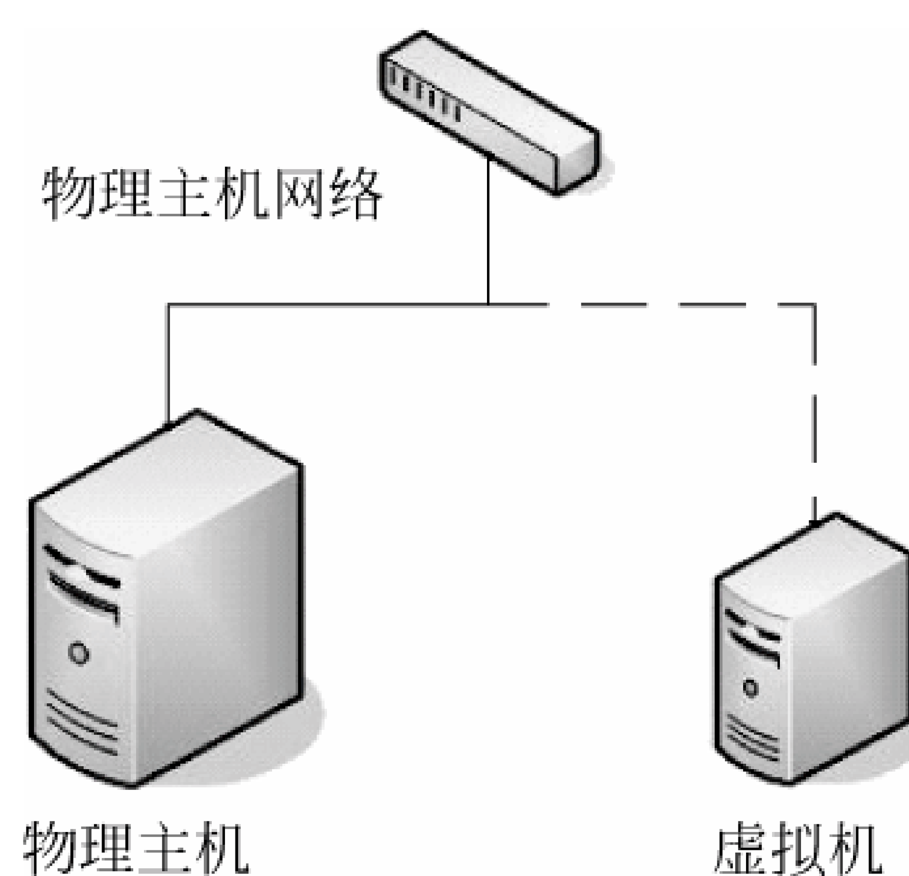


图 1.5　桥接方式

3．Host-only方式

使用 Host-only 方式时，虚拟机网络与物理机网络处于同

一位置，此时虚拟机和物理机相当于连接在同一交换机中的两台主机，如图 1.6 所示。

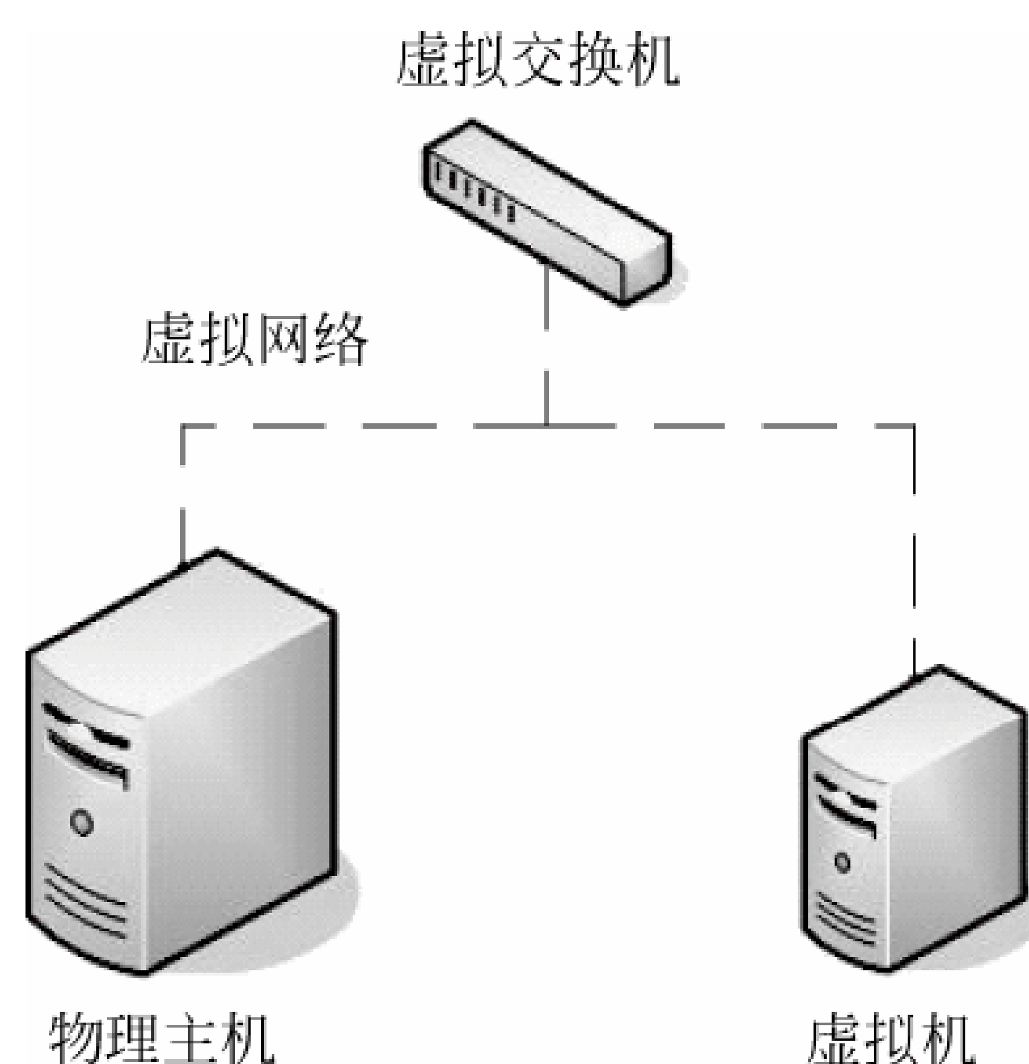


图 1.6　Host-only 方式

从图 1.6 中可以看出，物理主机与虚拟机连接到同一个交换机上，此时物理主机使用的网络连接是 VMnet1（必须保证网络连接 VMnet1 可用）。

Host-only 方式同 NAT 一样，虚拟机将会自动获得 IP 地址信息（无法获得网关地址）。虽然 Host-only 方式可以自动获得 IP 地址信息，但却无法使用主机网络。此时可以使用桥接或共享网络的方法将物理主机上的网络与 VMnet1 连接在一起，桥接和共享网络与 NAT 方式类似。不同之处在于，使用 Host-only 方式时远程主机可以使用端口映射（PNAT）的方法访问虚拟机。这种方式一般用于特殊环境，例如宽带拨号、VPN、无网络环境等。

提示： 由于 Host-only 需要自动获取 IP 地址，因此必须保证系统服务 VMware DHCP Service 可用。

4．虚拟网络

除了前面介绍的几种方法之外，虚拟机还提供了一些虚拟网络 VMnet2-7 和 VMnet9。使用这些网络时，运行在不同物理主机上的虚拟主机可以互相连接，但前提条件是这些物理机必须处于同一网络位置，且使用相同的自定义网络，如图 1.7 所示。

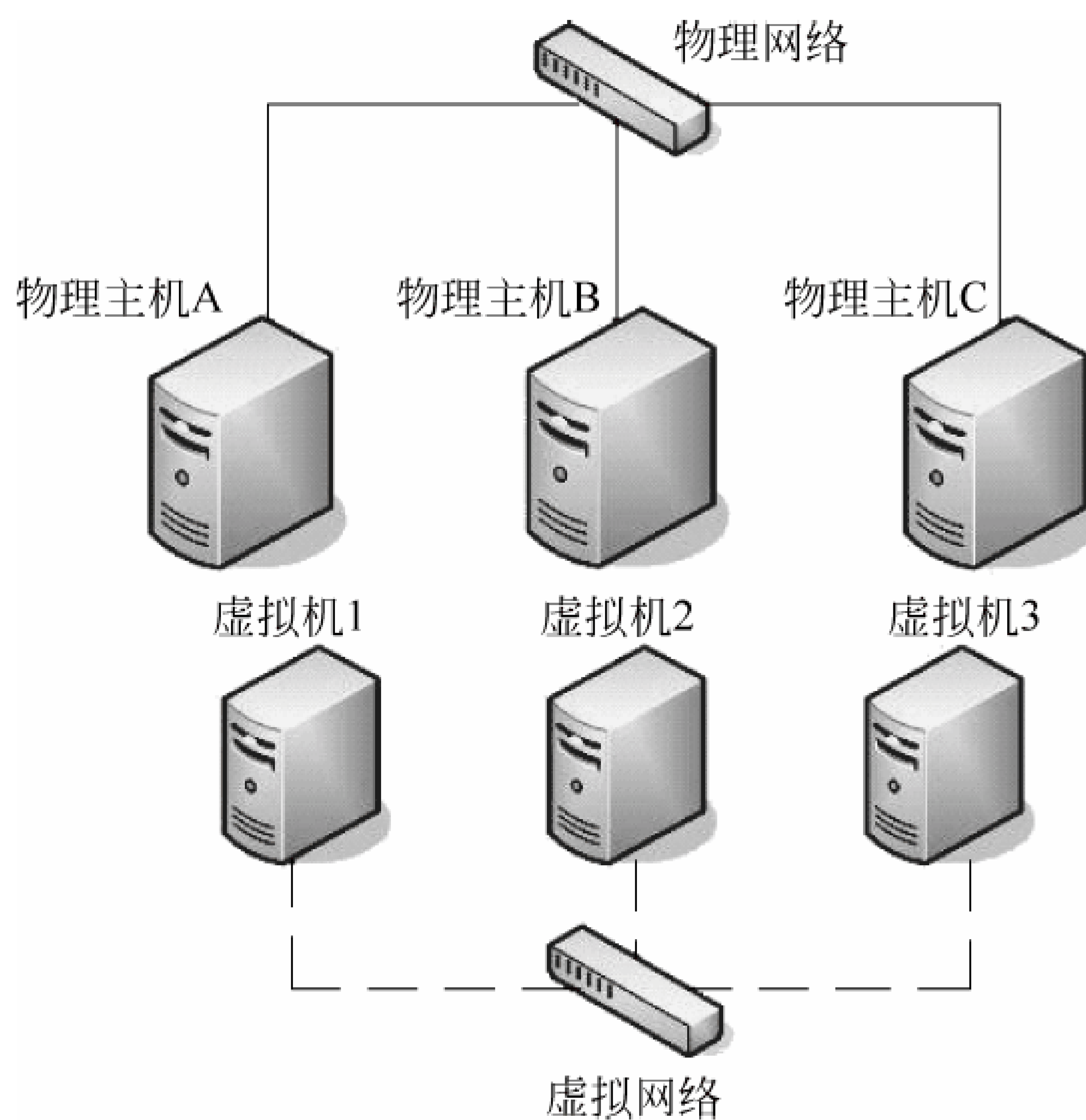


图 1.7　虚拟网络

在图 1.7 中，处于同一物理网络的物理主机 A、B、C 中，包含了虚拟机 1、2、3。当虚拟机 1、2、3 使用同一个虚拟网络时（例如都使用 VMnet2），这些位于不物理主机的虚拟机将处于同一网络位置。虚拟网络通常用于模拟大型的网络，提供更复杂网络环境（例如可用来进行大型网络实验）。

注意： 在本小节中使用了“同一网络位置”一词，此词的含义是两主机使用同一网段进行连接（可以互相通信），也可以理解为此时两主机连接到同一个交换机（且使用相同的 VLAN）。

技巧： 当物理主机上存在多个网络连接时，虚拟机可能会出现桥接网络混乱的现象。这时单击开始菜单→所有程序→VMware→Virtual Network Editor，在弹出的对话框中选择正确的桥接关系即可。

1.3.4　VMware Workstation 的使用技巧

VMware Workstation 有许多使用技巧，本小节将简单介绍几个常用的技巧。

- 使用组合键：由于组合键 Ctrl+Alt+Del 与系统冲突，因此如果需要对虚拟机使用此组合键，应该使用 Ctrl+Alt+Insert 代替。
- 电源键和重启键：虚拟机的电源键与物理主机的电源没有任何关系，如果需要虚拟机使用电源键和重启键，应该使用 VMware 工具栏中的电源、重启按钮。
- 键盘和鼠标：虚拟机可以使用物理主机的键盘和鼠标，单击虚拟机运行界面即可让虚拟机捕获当前系统的键盘和鼠标。要将键盘和鼠标切换回物理主机，可以使用快捷键 Ctrl+Alt。
- 虚拟机光驱：虚拟机可以使用物理光驱，也可以使用光盘的 ISO 镜像作为光驱。
- 虚拟机硬盘：添加虚拟机硬盘时，通常推荐使用 SCSI 硬盘，并且虚拟硬盘通常以每个文件 2GB 或多文件形式存储（以便于更好的兼容和迁移）。如果有特殊需要，也可以直接使用物理机的一个分区。

VMware Workstation 提供了一个良好的学习环境，但设置相对较复杂，初学者可以阅读相关文章获得帮助。

1.3.5　VMware ESX Server 简介

VMware ESX Server 是 VMware 公司为企业用户提供的虚拟化产品，与桌面解决方案不同的是，VMware ESX Server 自身就是一个操作系统（也是一个 Linux 操作系统）。VMware ESX Server 只能安装在服务器上，对服务器硬件要求较高，并且只能提供有限的硬件支持。虽然如此，但 VMware ESX Server 的功能十分强大，目前已被广泛应用于各大企业中。本小节将介绍 VMware ESX Server 及其优势。

VMware ESX Server 与桌面虚拟化产品 VMware Workstation 的另一个不同之处在于使用方法不同。在安装有 VMware ESX Server 操作系统的物理主机上，仅提供修改密码、设置 IP 地址等非常有限的几个功能。如果需要对虚拟机进行操作，只能使用一个名为 VMware Infrastructure Client 的软件连接到服务器上，然后才能对虚拟机进行操作。VMware Infrastructure Client 的运行界面如图 1.8 所示。

VMware ESX Server 具备以下功能和特性。

- 操作虚拟机、查看日志、管理所有虚拟机等任务，只需要安装一个客户端即可，使用起来非常方便。
- 虚拟机使用的存储设备不仅可以是服务器自身的硬盘，还可以使用 iSCSI、NFS 等网络存储。
- 使用 iSCSI、NFS 等网络存储时，可以实现故障迁移，即当服务器出现故障时，使用网络存储迅速恢复服务。
- VMware ESX Server 不仅可以为虚拟机分配内存，还可以为虚拟机分配计算资源，非常有利于控制虚拟机使用的资源。

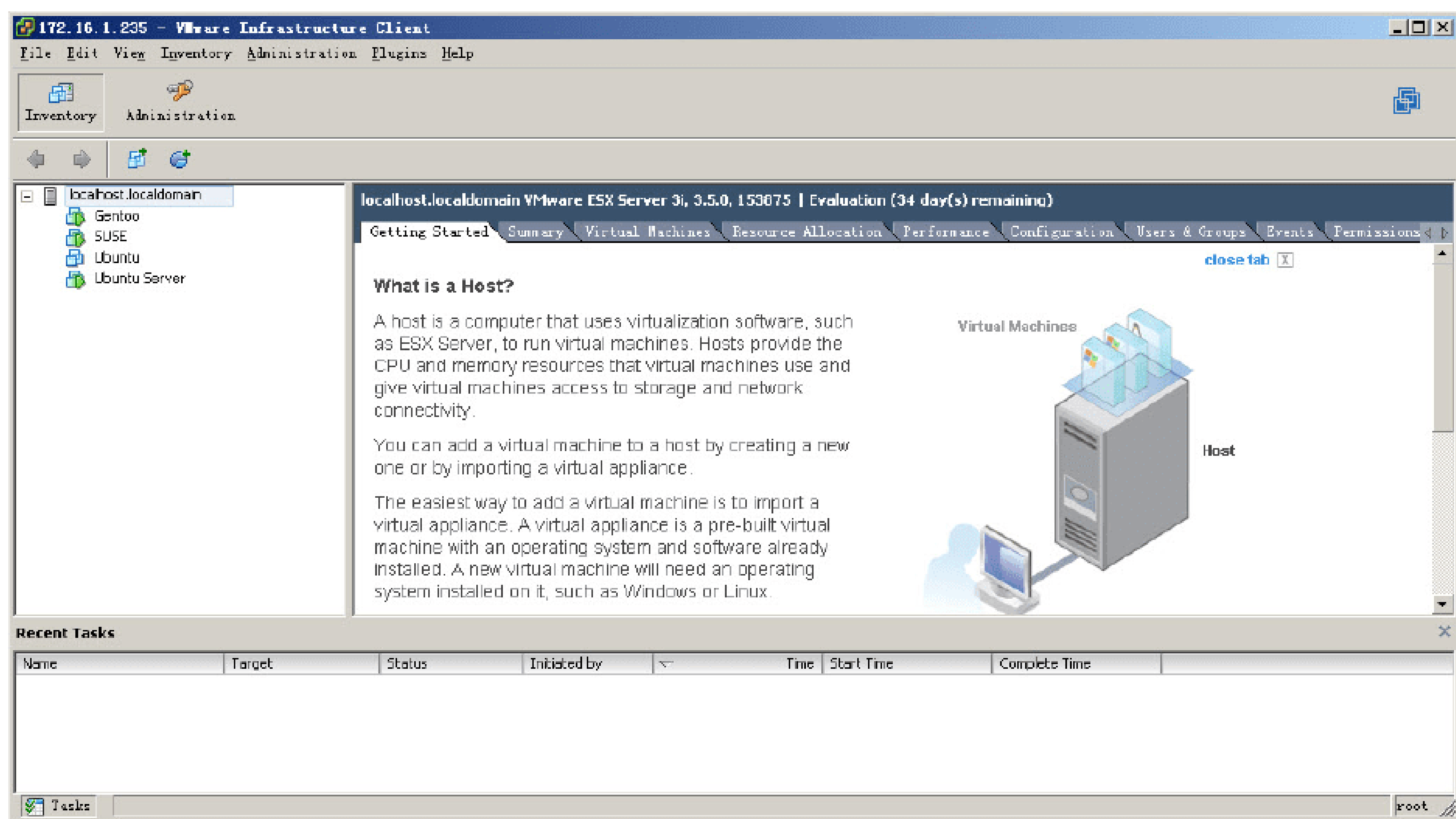


图 1.8　VMware Infrastructure Client 运行界面

作为企业级虚拟化解决方案，VMware ESX Server 使用非常广泛，感兴趣的读者可以阅读相关资料了解并学习。

1.4　Linux 安装过程

讲解了 Linux 系统的优势及学习 Linux 的工具之后，下面将要迈出走向 Linux 系统的第一步：将 Linux 系统安装到主机中。本节将简单讲解如何安装 Linux 系统。

说明： 本书大部分内容将以 Red Hat 公司的企业版 Red Hat Enterprise Linux 5.3（以下简称为 RHEL5.3）为例进行讲解。

1.4.1　安装前的准备工作

与 Windows 系统一样，在安装 Linux 系统之前，还需要做一些准备工作。这些准备工作可能包括：硬件兼容性检查、合适的安装介质、磁盘空间规划等。

（1）兼容性检查

在安装 Linux 系统之前，应该查阅发行版官方网站的兼容列表，检查发行版是否兼容你的硬件。以下情况通常会出现兼容性问题。

- 较新的 Linux 版本可能无法支持十分古老的硬件（一般为 10 年以上）。
- 在最新硬件平台上安装较老的 Linux 发行版时，可能会出现无法识别硬件的问题。

除此之外，一些比较特殊的硬件（例如显卡、阵列卡、硬件控制模块等），可能需要为其准备驱动程序，或阅读产品的安装说明等。

（2）合适的安装介质

RHEL5.3 可以使用光盘引导安装，如果主板支持，也可以使用 U 盘等设备引导安装。使用非光盘介质安装时，可能需要从其他位置读取软件包，通常支持从光盘、硬盘和文件

服务器中读取。

在安装之前还需要仔细规划硬盘分区方案，并将硬件设备都连接到目标主机，例如硬盘、网络接口卡等。

1.4.2　创建虚拟机并使用光盘引导

本书将在 VMware Workstation 虚拟机中介绍如何安装 Linux，首先要进行的操作是创建虚拟机并修改虚拟机的引导设备。本小节将简单介绍如何创建虚拟机及使用光盘引导虚拟机。

1．在VMware Workstation中创建虚拟机

（1）在 VMware Workstation 中创建虚拟机有两个方法：单击 Home 标签中的 New Virtual Machine；单击菜单栏中的 File→New→Virtual Machine。无论使用哪种方法都会弹出创建虚拟机向导，如图 1.9 所示。从中可以看出创建虚拟机有两种方式：第 1 种方式是 Typical（典型），这种方式建立的虚拟机许多地方都使用默认设置（例如使用 SCSI 硬盘、NAT 网络连接等）；第 2 种方式是 Custom（自定义），在这种方式下用户可以选择创建虚拟机兼容的版本、硬盘接口类型等。如果没有特殊需要，通常建议使用典型方式创建虚拟机。

（2）选择创建虚拟机方式后，单击 Next 按钮即可进入选择安装介质界面，如图 1.10 所示。

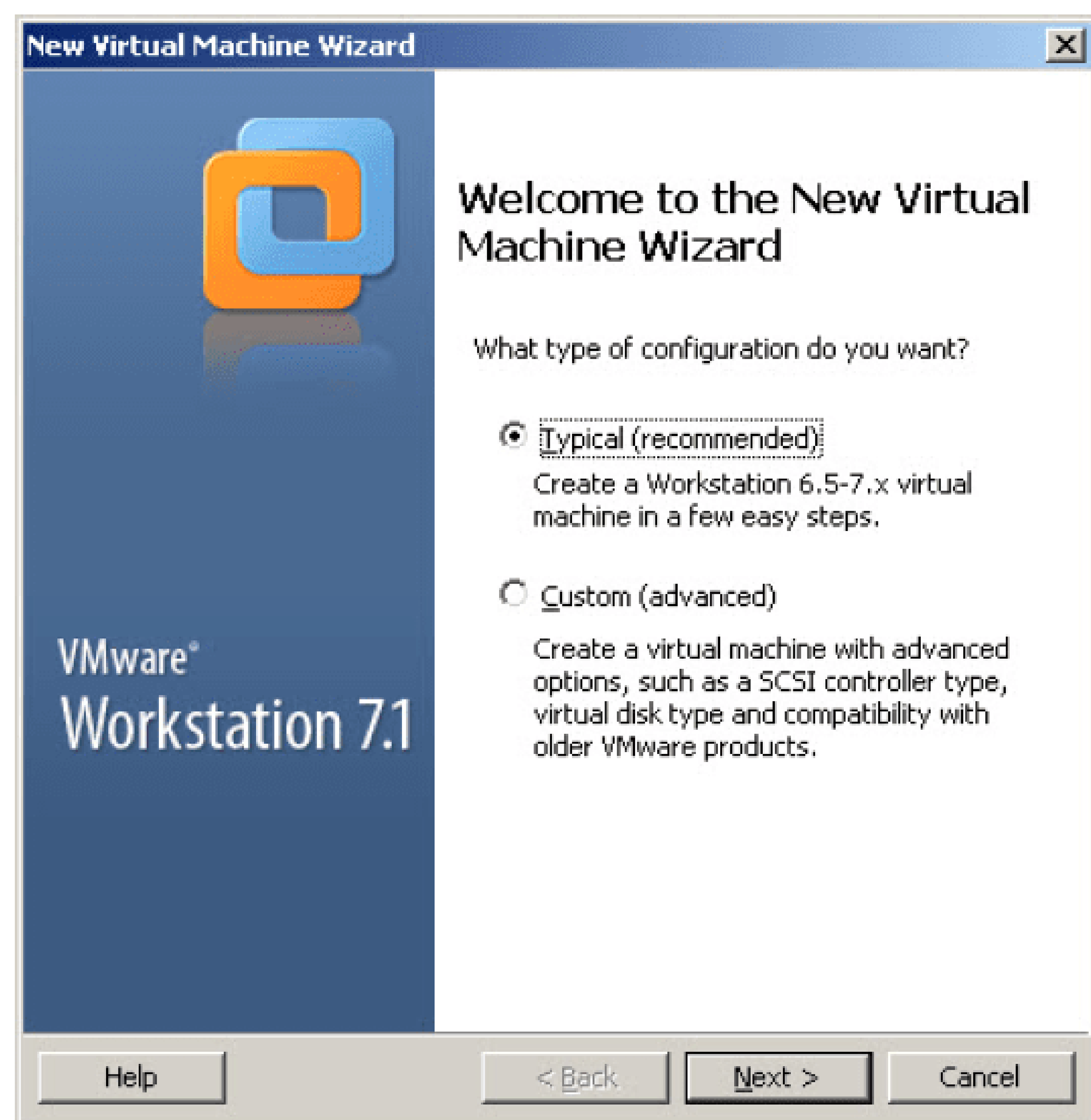


图 1.9　创建虚拟机向导

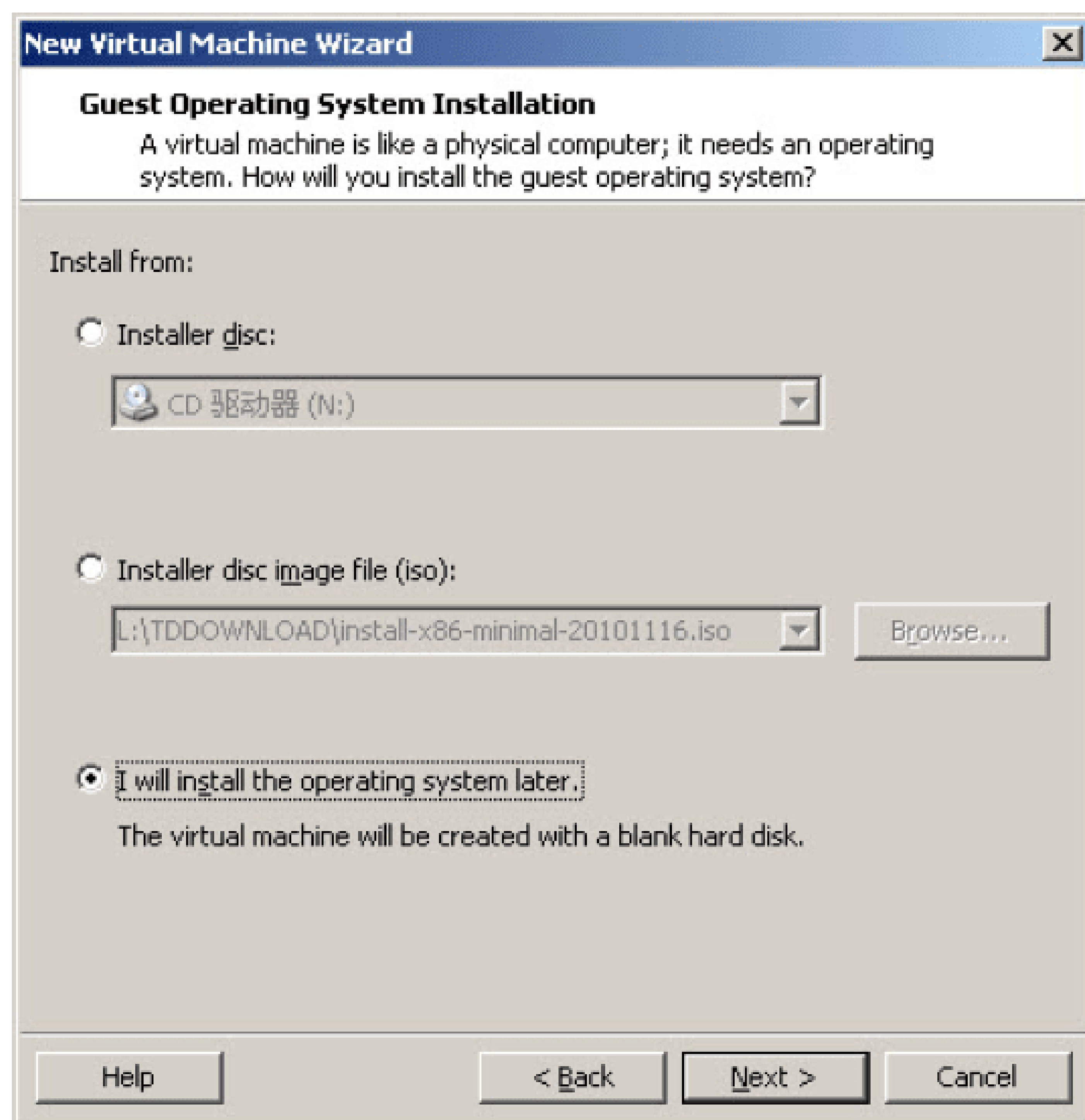


图 1.10　选择安装介质界面

安装操作系统步骤要求用户选择安装介质，用户可以在 Installer disc 下拉列表框中选择装有安装光盘的驱动器，也可以在 Installer disc image file(iso)下拉列表框中选择光盘镜像文件。本例中选择第 3 个选项 I will install the operating system later（稍后安装操作系统）。

提示：如果在上面的步骤中选择了合适的安装介质，虚拟机创建完成后，VMware Workstation 将会使用内置的默认设置立即开始操作系统的安装过程。由于使用

VMware Workstation 的默认设置，安装的软件包可能并不完全（例如不会安装 xinetd 老式服务集、Vim 编辑器等），因此通常不建议使用这种方式安装操作系统。

（3）选择安装介质后，单击 Next 按钮进入选择操作系统界面，如图 1.11 所示。从选择操作系统界面中可以选择虚拟机的操作系统类型，本例中先选中 Guest operating system 选项组中的 Linux 单选按钮，然后在 Version 下拉列表中选择“Red Hat Enterprise Linux 5”。

注意：如果选择的操作系统与安装的操作系统不匹配，可能会出现无法正常安装、使用虚拟机的现象。

（4）完成虚拟机操作系统选择后，单击 Next 按钮即可进入命名虚拟机界面，如图 1.12 所示。

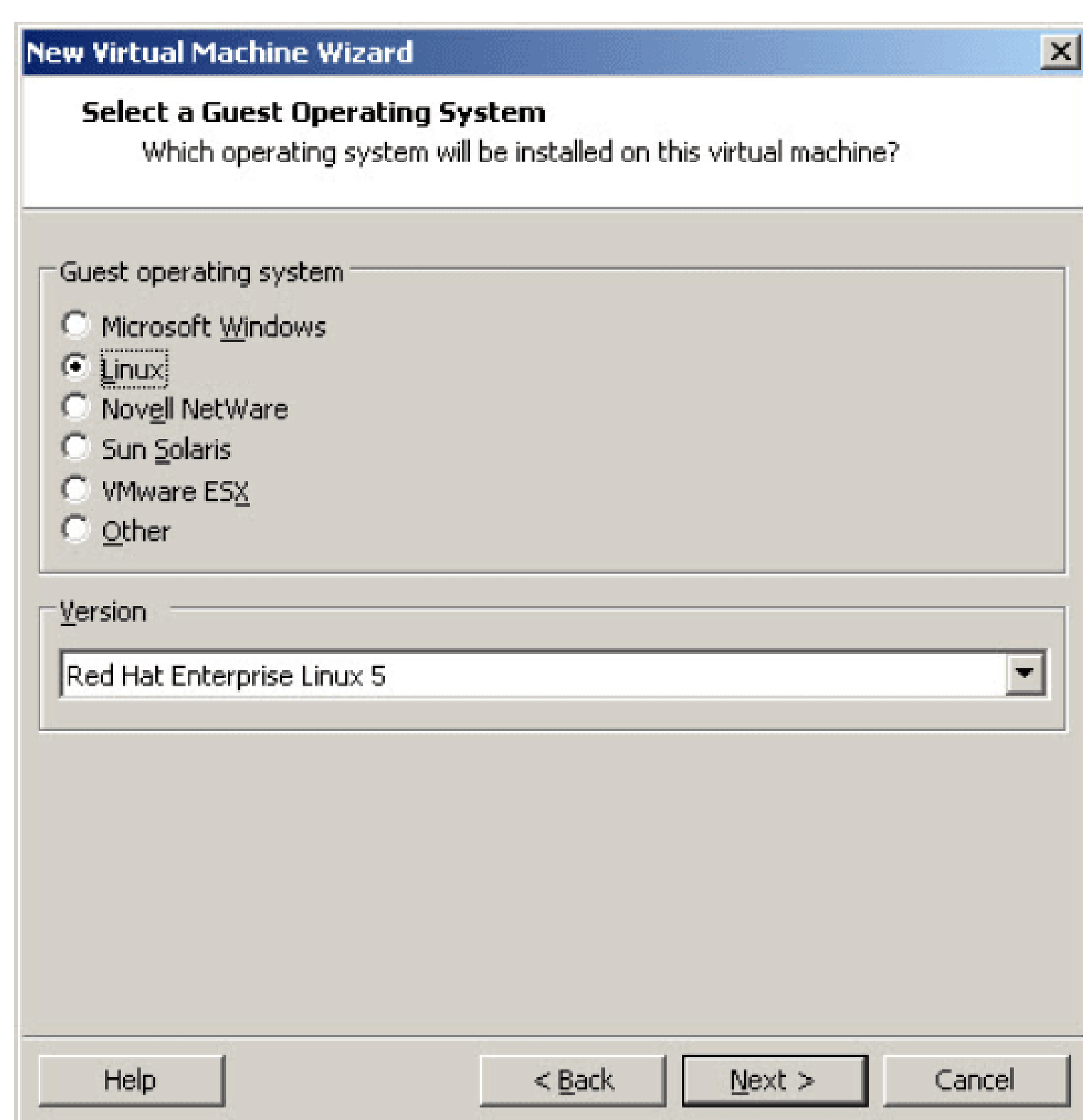


图 1.11　选择操作系统界面

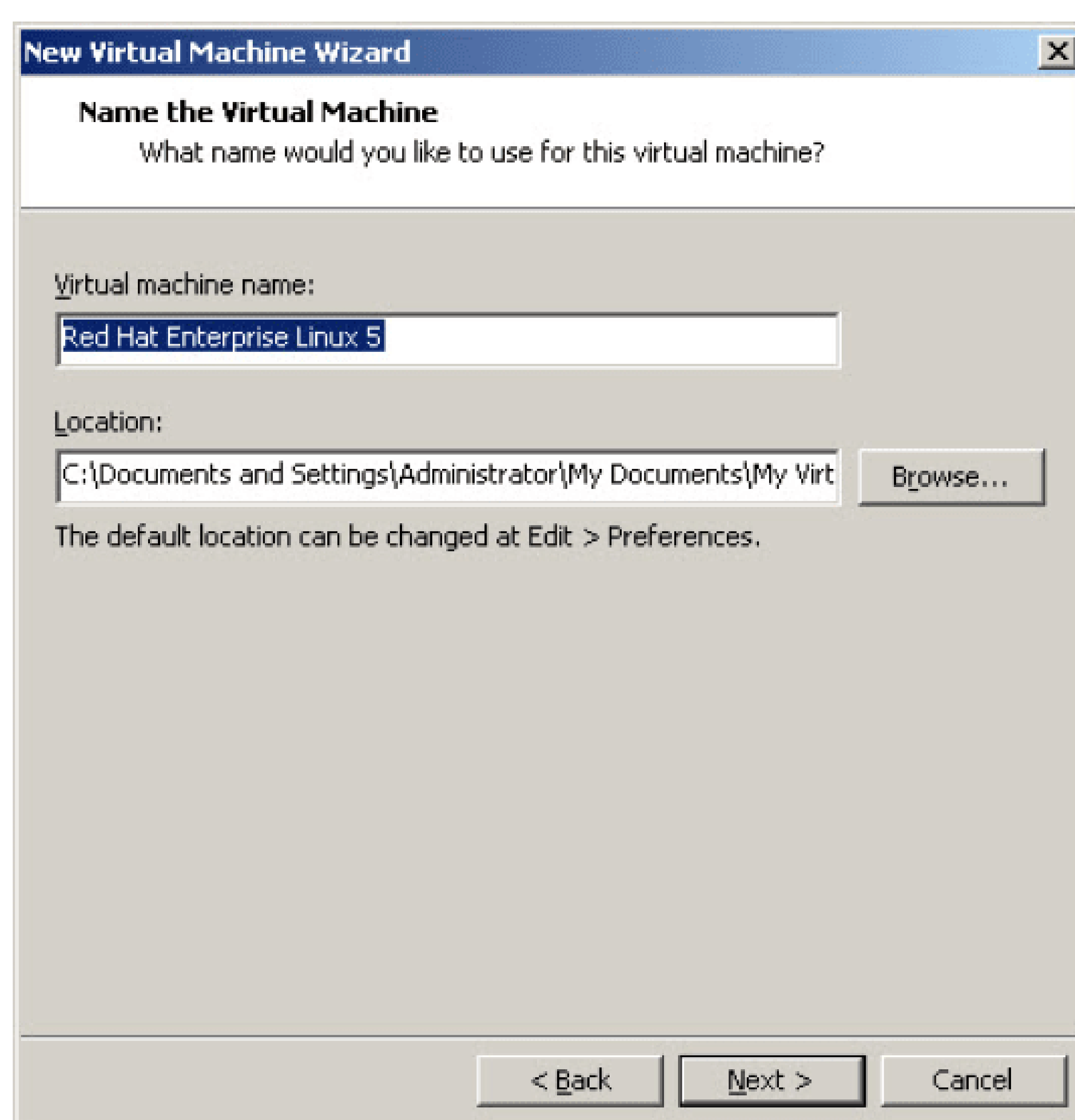


图 1.12　命名虚拟机

在命名虚拟机界面中，需要为新建的虚拟机建立一个容易识别的名称。如果建立的虚拟机主要用于学习，通常建议按虚拟机的用途命名；如果是实际应用环境，通常虚拟机名称中应该包含操作系统类型、虚拟机用途和 IP 地址等信息，以便于管理员识别。用户可以按实际情况在 Virtual machine name 文本框中输入虚拟机的名称，并在 Location 选择框中选择虚拟机保存的位置。

（5）为虚拟机命名后单击 Next 按钮，即可进入定制虚拟机磁盘界面，如图 1.13 所示。

用户可以在定制虚拟机磁盘界面中定制虚拟机磁盘大小，以及磁盘文件在计算中保存的方式。本例中使用的虚拟机硬盘大小为 20GB，磁盘文件的保存方式为“Split virtual disk into multiple files”（将虚拟磁盘文件保存在多个文件中）。

（6）完成上述步骤之后，将会进入到准备创建虚拟机界面，如图 1.14 所示。

在准备创建虚拟机界面中，用户可以查看即将创建的虚拟机的各项设置，也可以单击 Customize Hardware 按钮，在弹出的 Hardware 对话框中修改虚拟机的硬件设置（例如修改虚拟机内存大小、CPU 核心数量等）。用户确认虚拟机硬件设置正确之后，单击 Finish 按钮即可完成虚拟机的创建。

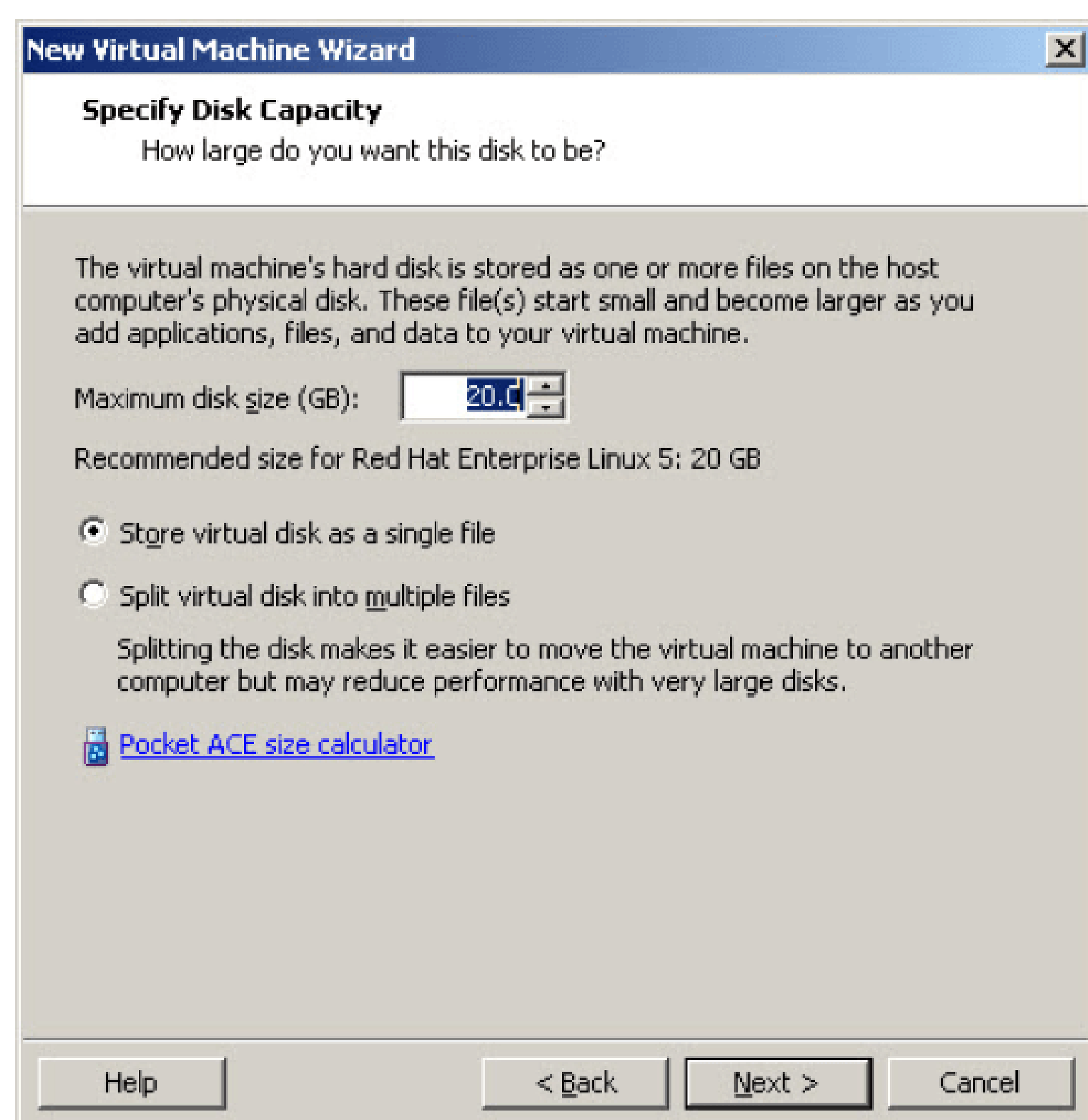


图 1.13　定制虚拟机磁盘界面

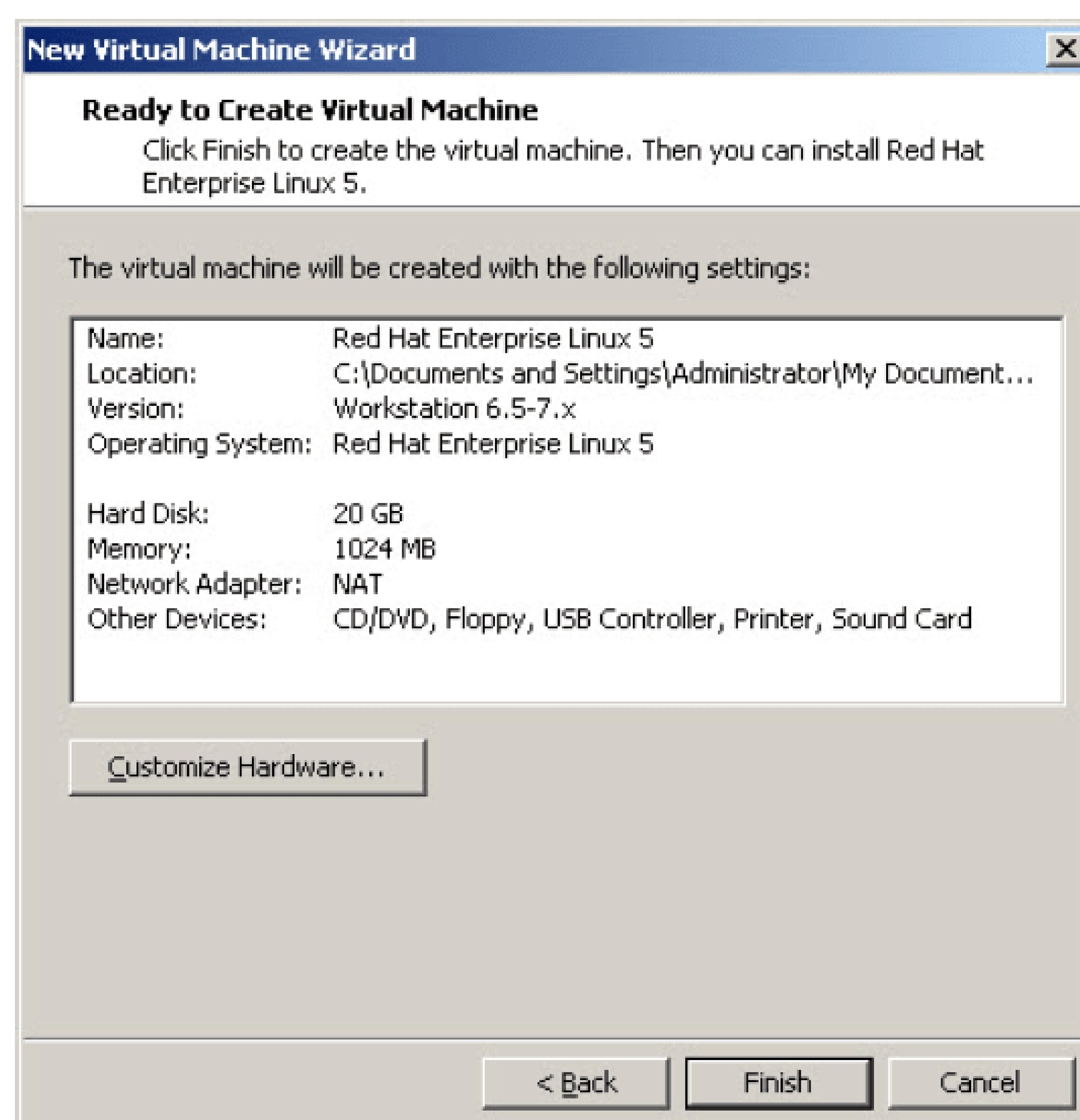


图 1.14　准备创建虚拟机界面

2. 修改虚拟机引导设备

创建完虚拟机之后，就可以插入安装光盘并开始系统安装过程了。

（1）插入安装光盘

要在虚拟机中插入安装光盘，可以单击虚拟机标签中的“Edit Virtual Machine settings”，此时将弹出 Virtual Machine Settings（虚拟机设置）对话框，如图 1.15 所示。

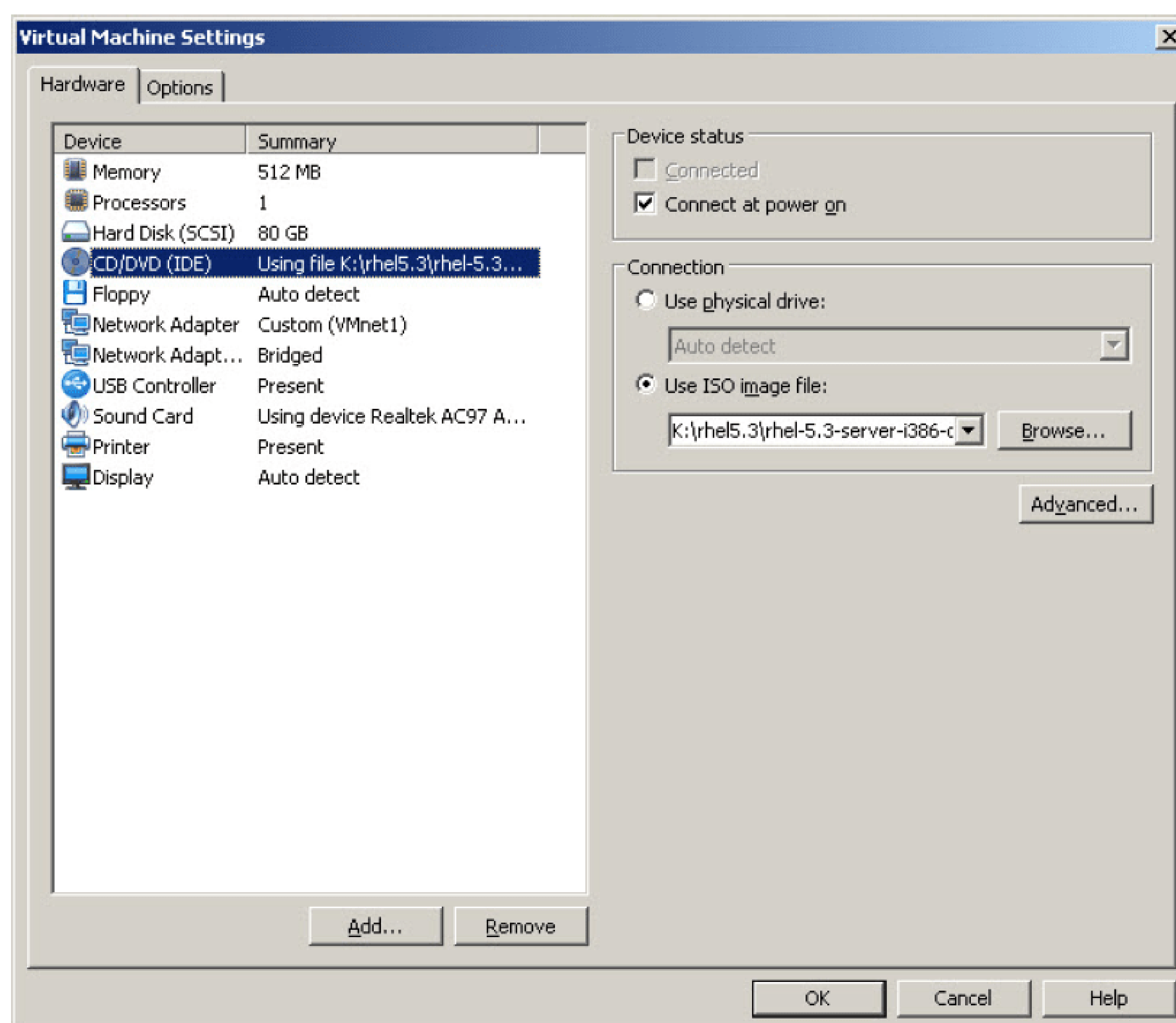


图 1.15　修改虚拟机设置

在 Virtual Machine Settings 对话框中单击“CD/DVD（IDE）”，之后就可以在对话框右侧设置虚拟机要使用的安装光盘。按实际需要选择“Use physical drive”（使用物理设备时需要将安装光盘放入相应的驱动器中）或“Use ISO image file”（使用 ISO 光盘镜像文件时需要指定文件的具体位置），完成后单击 OK 按钮即可。

（2）修改虚拟机引导设备

由于 VMware Workstation 的新版本支持虚拟机自动查找引导功能，因此可以不必修改虚拟机的引导设备。如果读者使用的是较早的版本，可以单击 VMware Workstation 工具栏中的 Power On 按钮接通虚拟机电源，启动时按 F2 键进入 BIOS 设置，在 Boot 菜单中选择优先使用光驱进行引导，如图 1.16 所示。

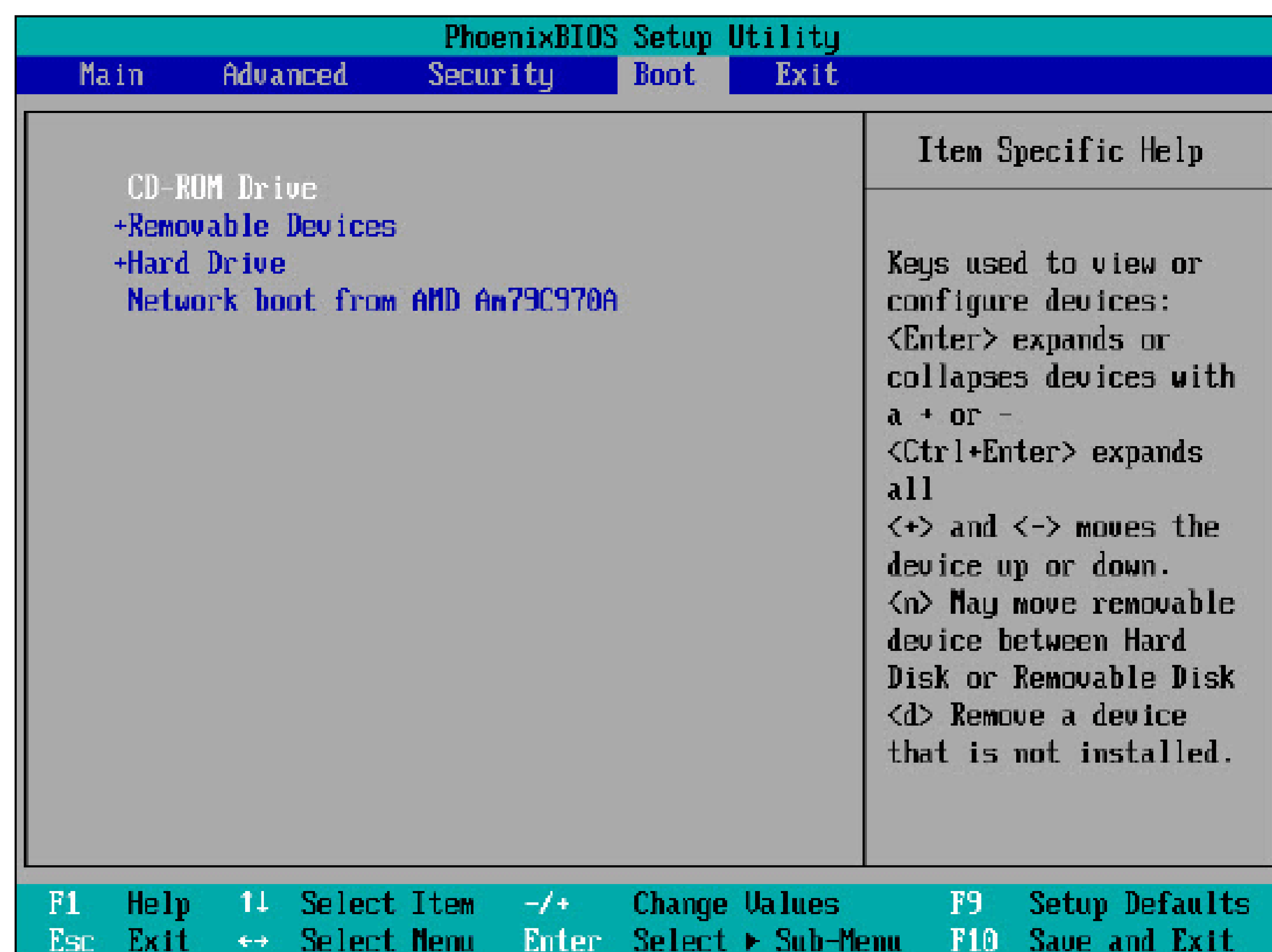


图 1.16　修改优先引导设备为光盘驱动器

注意：启动虚拟机之后，需要先在虚拟机控制台（即虚拟机启动后的显示界面）中单击鼠标左键，让虚拟机捕捉到物理机的鼠标和键盘才能对虚拟机进行操作。

设置完引导设备后，按 F10 键保存并退出 BIOS 设置，此时虚拟机将会重新启动并使用光盘引导系统。

1.4.3　安装模式和光盘检测

计算机从光盘引导成功之后，安装程序将会停留在安装模式选择界面，如图 1.17 所示。

（1）从选择界面中可以看出，有两种安装模式可供选择：图形模式和文本模式。此时可以进行的操作如下。

- 按 Enter 键将直接使用图形安装模式。
- 如果需要使用文本安装模式，此时在冒号提示符后面输入“linux text”，按 Enter 键即可进入文本安装模式。文本安装模式的优点是节省时间，但文本安装模式不如图形安装模式直观。
- 按 F2～F5 键可以查看其他模式，以及模式使用的命令等。



图 1.17　启动安装程序

提示：建议初学者使用图形模式安装，待熟悉安装过程之后再使用文本模式安装，以节省安装时间。

在本例中按 Enter 键选择图形模式安装 Linux，此时安装程序将会检查硬件设备并加载安装介质检查工具。

（2）完成后会询问用户是否对安装介质进行完整性检查，如图 1.18 所示。

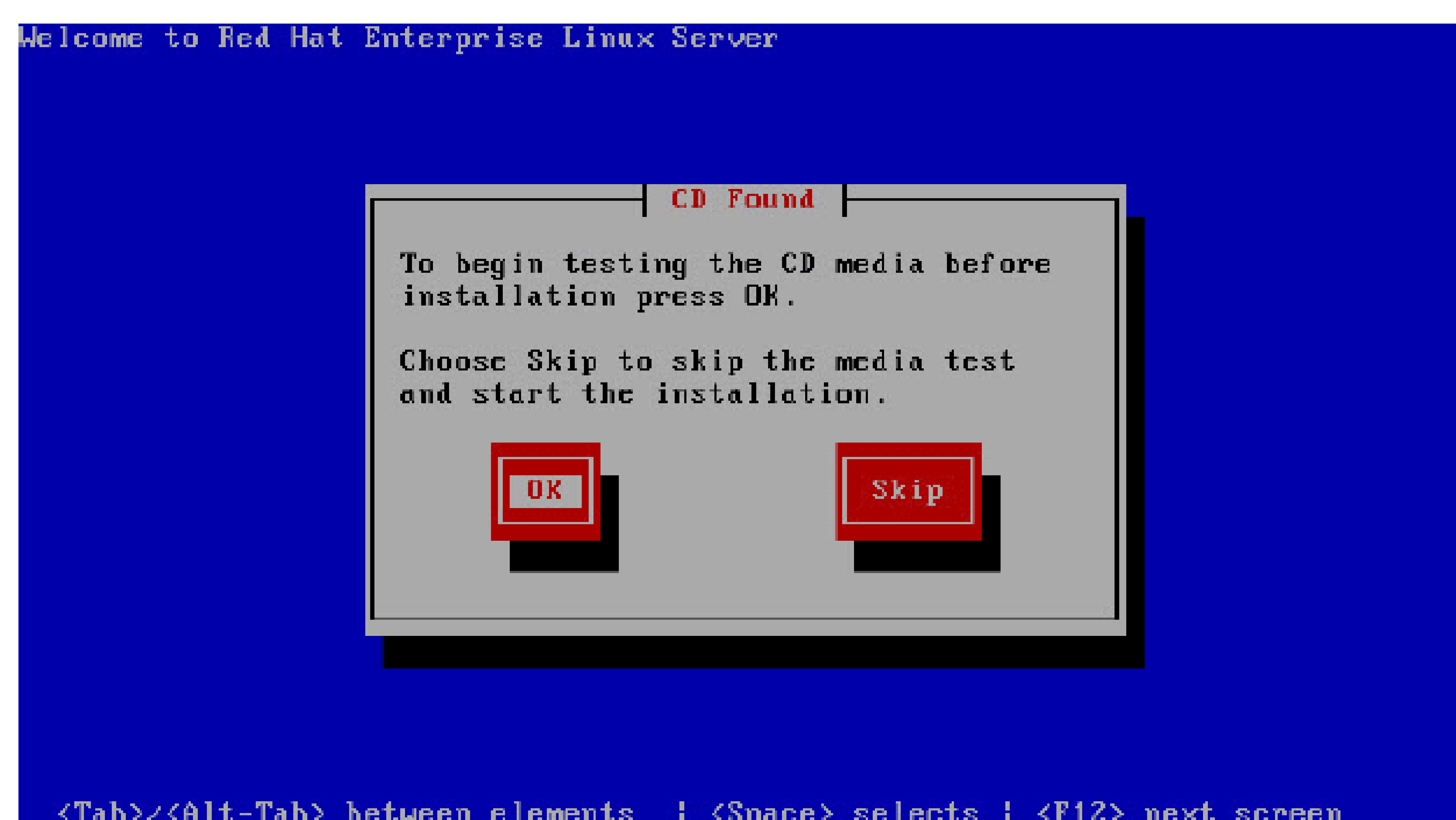


图 1.18　检查光盘完整性

有时从互联网下载的光盘镜像可能会存在错误，也可能由于光盘质量原因，光盘中某些文件无法读取等。这些都有可能导致安装程序无法读取某个软件包，进而导致整个安装计划失败。完整性检测将检查光盘中的每个文件是否可以正常读取，避免光盘存在错误导致安装失败。

如果需要对安装光盘进行完整性检测，此时可以选中 OK 按钮，按下 Enter 或空格键。也可以使用 Tab 键切换到 Skip，然后按 Enter 键或空格键，跳过光盘检测步骤。

提示：如果使用 CD 光盘作为安装介质，光盘完整性检查将会检查所有光盘的完整性（CD 安装光盘一共 4 张），按提示插入光盘即可完成完整性检查。

1.4.4　图形安装环境配置

完成光盘检查步骤之后，安装程序会加载图形界面进行后续安装过程。安装程序启动图形模式后会停留在欢迎界面，如图 1.19 所示。



图 1.19　图形模式的欢迎界面

🔔注意：如果主机硬件不足以启动图形安装模式（常见的原因是内存太小），安装程序会自动启动文本安装模式。

（1）单击 Next（下一步）按钮，进入选择语言界面，此时可以在语言选择列表中选择安装过程使用的语言。本例中选择“English”作为安装过程使用的语言，如图 1.20 所示。



图 1.20　选择安装过程使用的语言

（2）选择安装程序使用的语言后，单击 Next 按钮，进入选择键盘界面，如图 1.21 所示。



图 1.21　选择键盘

除非使用了其他的键盘，否则一般情况下都选择使用美国英语式键盘（列表中选择“U.S.English”）。

（3）完成键盘选择后单击 Next 按钮，安装程序会提示输入安装号，如图 1.22 所示。

在 RHEL5.3 中，安装号用于区分用户安装的操作系统版本，输入的安装号不同，可能安装完成后是服务器版（Server）、客户端版（Desktop）等。不同的 RHEL 版本默认安装的软件有所不同，用户也可以不输入安装号自行定制需要安装的软件。

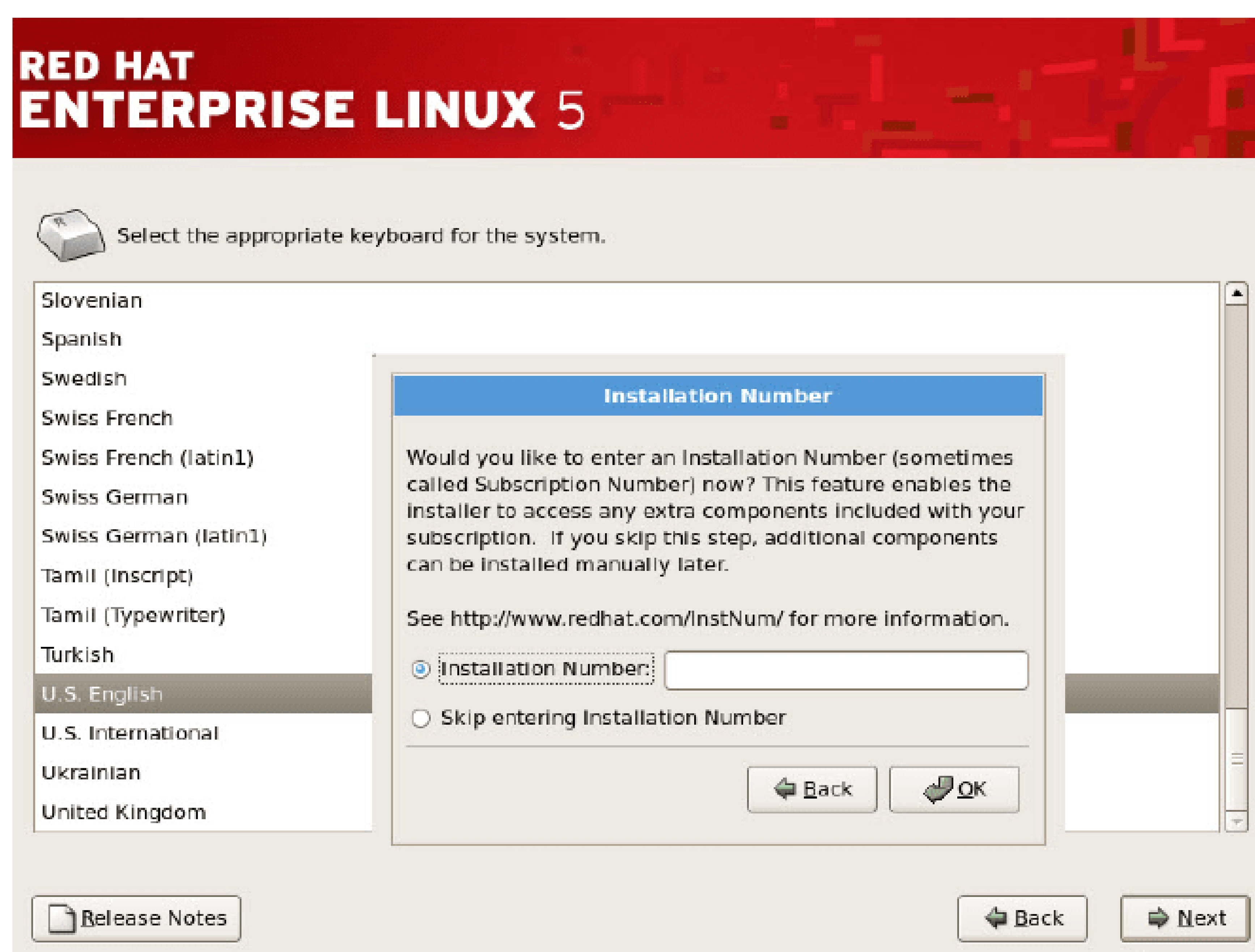


图 1.22　输入安装号码

1.4.5　磁盘分区

输入或跳过安装号步骤之后，安装程序将会进入硬盘分区步骤，本小节将介绍如何在安装过程中进行磁盘分区。由于虚拟机中使用的是新硬盘，因此安装程序会提示没有找到硬盘中的分区表，并询问是否需要初始化磁盘，详细的步骤如下所示。

注意： 初始化磁盘会丢失磁盘上的所有数据，如果在真实计算机上操作，应该特别谨慎，以免损坏磁盘中的数据。

（1）选择初始化磁盘之后，安装程序将会停留在选择磁盘分区方式界面，系统会提示是否使用默认的分区方案，如图 1.23 所示。

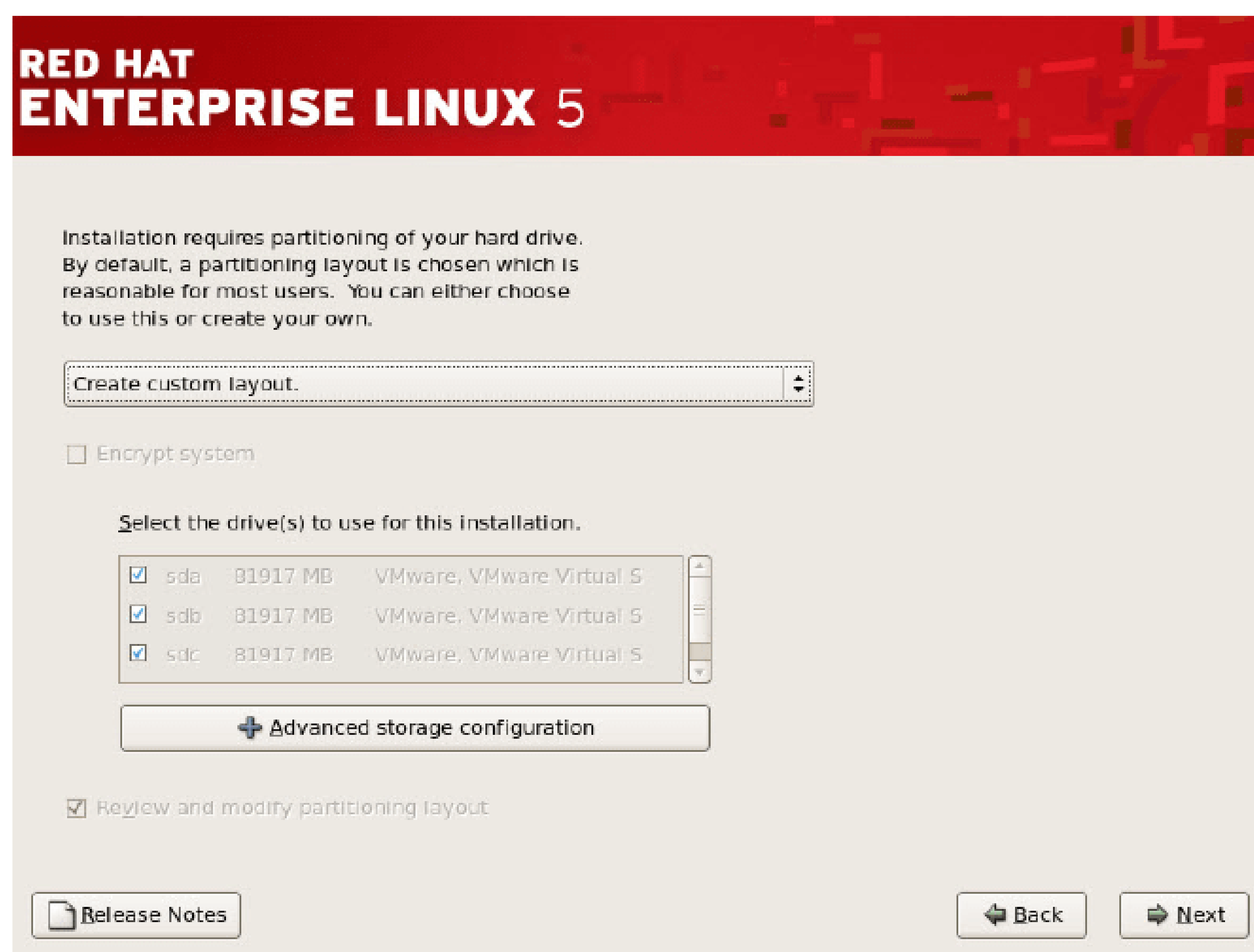


图 1.23　选择分区方式

通常建议初学者使用手动方式建立分区结构，以加深对挂载等概念的理解。本例中使用手动方式建立分区结构，在下拉列表中选择“Create custom layout”（创建自定义结构），然后单击 Next 按钮进入自定义分区界面，如图 1.24 所示。

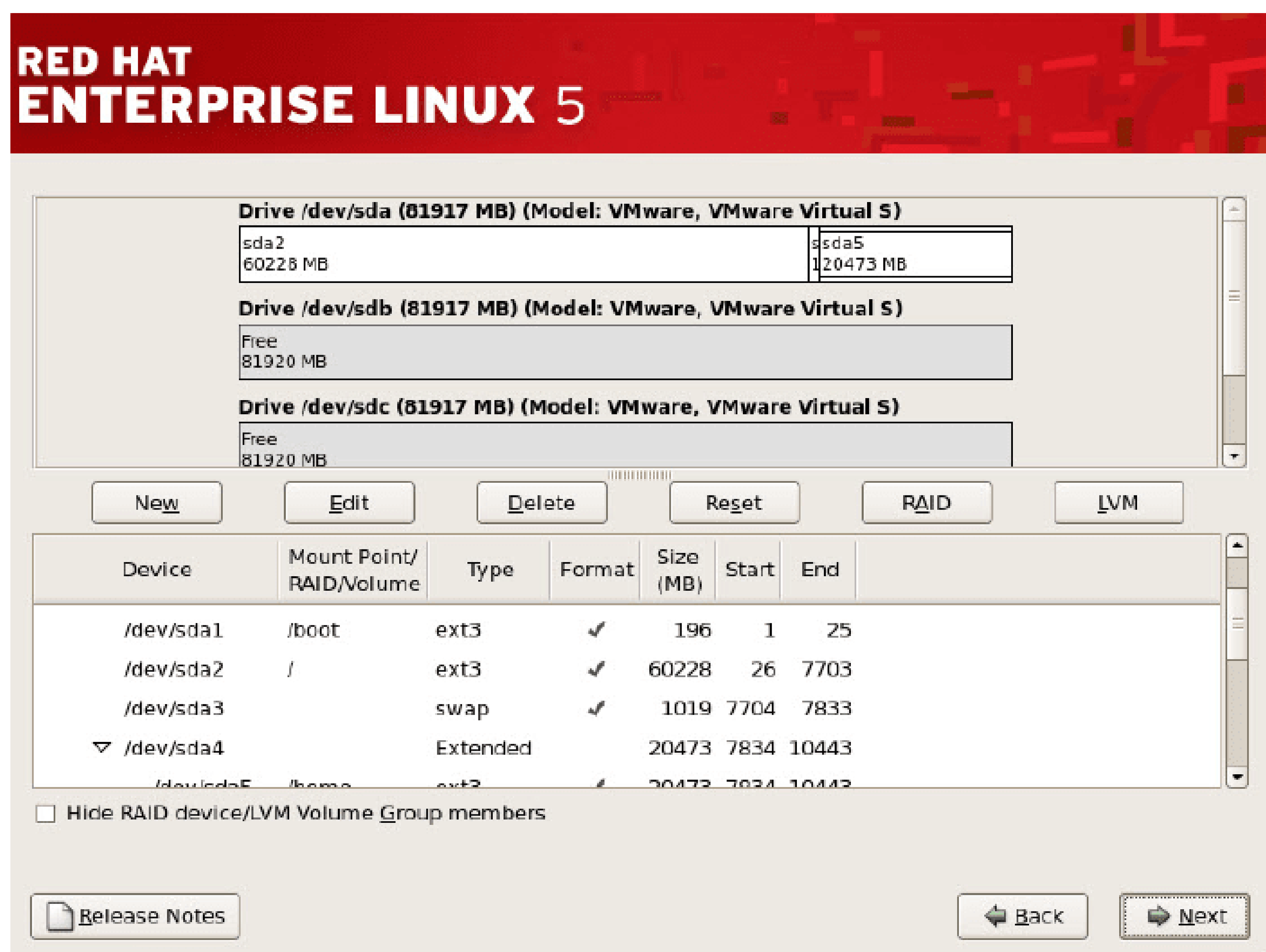


图 1.24　自定义分区界面

（2）开始动手建立分区结构之前，需要建立一个合适的分区方案，一个合适的分区方案应该注意以下内容。

❑ 尽量不要将系统安装在无冗余功能的 Raid 设备上。
❑ 如果系统中有多个硬盘，建议将系统安装在其中一个硬盘内，其他硬盘用来存放业务数据等。
❑ 如果当前系统用作数据中心，建议使用 LVM 建立分区结构，以便于日后进行在线式扩展。

对于初学者而言，一个最简单的分区方案是设置一个启动分区和一个交换分区，然后将其他空间全部挂载到根目录/下面。

小知识：为了加快系统读写虚拟内存时的速度，通常将虚拟内存设置为一个单独的分区，称为交换分区。通常为系统预留的交换分区大小应为物理内存的 2 倍，任何时候都应该考虑添加交换分区。

提示：/boot 为系统引导目录，用于存放内核文件、引导配置文件等，通常应该单独设置一个分区，其大小设置为 100MB 即可。/home、/root 是普通用户和 root 用户的家目录，应该视系统的实际使用情况设置分区大小。

注意：为服务器规划分区时，应该将系统数据、用户数据和业务数据等种类繁多的数据放置到不同的分区或磁盘中，以确保存储设备出现问题时，能够最大限度地保全数据。

（3）建立好分区方案后就可以开始磁盘分区了，以划分引导分区为例，单击 New 按钮，在弹出的对话框中输入（也可以在其后的列表中选择）挂载点（Mount Point）/boot。然后选择引导分区的文件系统类型（File System Type），建议选择文件系统的类型为 ext3。然后在 Allowable Drives 选择框中勾选需要操作的磁盘，并在 Additional Size Options 选项中指定分区大小，最后单击 OK 按钮即可建立引导分区。

添加完引导分区之后，还应该为系统添加一个交换分区。添加交换分区的过程与建立引导分区类似，不同的是交换分区不需要使用挂载点，并且交换分区的文件类型应该设置为 swap（也因此交换分区通常称为 swap 分区）。交换分区的大小通常为内存的 2 倍，但内存容量超过 1GB 时，建议将交换分区大小设置为内存大小的 1～1.5 倍。

（4）完成之后可以添加根分区和其他分区，方法与添加引导分区类似，逐一添加所有分区后，单击 Next 按钮即可完成分区操作。

提示：从以上步骤中可以看出，挂载就是将访问磁盘分区的接口设置为系统中的一个目录。

1.4.6　引导程序和网络配置

在完成磁盘分区之后，就进入为系统安装引导装载程序、配置网络连接步骤了。本小节将简单介绍安装过程中的引导程序设置和网络配置。

1．安装引导装载程序

完成分区操作后，安装程序将提示是否安装引导装载程序 GRUB，如图 1.25 所示。如果主机上还安装有其他操作系统，此时还需要为其他操作系统添加引导选项，通常此步骤保持默认即可。



图 1.25　安装引导装载程序

2．设置系统网络

安装完引导装载程序后，安装程序将会提示用户配置网络环境，如图 1.26 所示。

小知识：在 RHEL 中使用以太网的英文 Ethernet 的缩写 eth 标识以太网网络接口卡，按网络接口卡在主板插槽上的位置不同，分别命名为 eth0、eth1、eth2…等。

如果设置系统网络步骤保持默认，网络接口将试图从 DHCP 服务器上获取 IP 地址、

子网掩码、默认网关等初始化信息。如果需要手动设置网络，可以参考以下步骤。

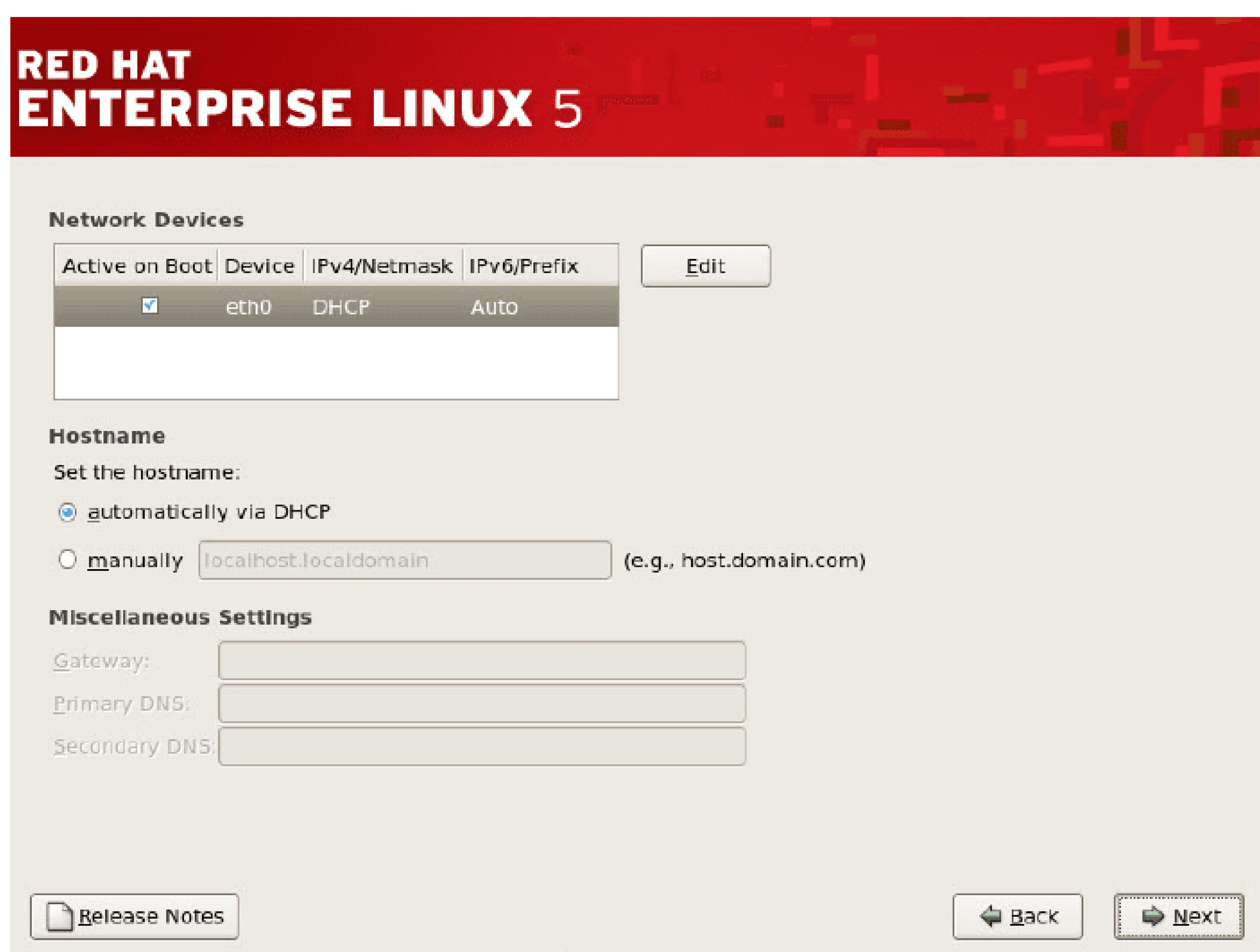


图 1.26　配置网络环境

（1）手动设置 IPv4 网络

如果网络中没有 DHCP 服务器，就需要手动设置网络。手动设置网络可以在 Network Devices（网络设备）选择框中选中相应的网络接口设备，单击 Edit 按钮。如果是配置 IPv4 网络，就在弹出的对话框中勾选“Enable IPv4 support”（开启 IPv4 支持）和“Manual configuration”（手动配置），并在 IP Address 和 Prefix（Netmask）文本框中输入 IPv4 地址和掩码，单击 OK 按钮即可设置完成。

（2）手动设置 IPv6 网络

手动设置 IPv6 网络需要在 Network Devices 选择框中选择相应的网络接口设备，并单击 Edit 按钮，在弹出的对话框中勾选“Enable IPv6 support”（开启 IPv6 支持）。IPv6 设置网络接口有如下两种方式。

- 自动获取：IPv6 可以使用自动邻居发现协议和 DHCP 两种方式自动配置网络。配置时按需要勾选“Automatic neighbor discovery”和“Dynamic IP configuration (DHCPv6)”即可。
- 手动设置网络：如果要手动配置 IPv6 网络，则勾选“Manual configuration”（手动配置），在其后的 IP Address 和 Prefix 文本框中输入 IP 地址和掩码，单击 OK 按钮即可完成设置。

（3）手动设置主机名称

设置主机名称可以在网络配置界面中的 Set the hostname:项中勾选“manual”，并输入需要使用的主机名。

（4）设置杂项

如果手动设置网络连接，还需要设置默认网关、主要和次要 DNS 服务器等信息，这些设置都在 Miscellaneous Settings（杂项设置）项中，按需要设置即可。

提示： 在虚拟机中安装 RHEL5.3 时，如果网络连接使用 NAT 方式，VMware 会使用 DHCP 自动为虚拟机分配 IP 地址和网关等信息，因此网络设置应该保持默认值。

1.4.7　设置时区和根用户密码

为保证系统有一个正确的时间和一个安全的使用环境，在安装时还需要设置时区和 root 用户的密码。

1．设置时区

完成网络环境的配置之后，系统将会提示用户设置时区，如图 1.27 所示。



图 1.27　设置时区

本例中选择 Asia/Shanghai 作为当前时区，实际操作时按需要进行选择即可。

注意：如果时区设置错误，可能会出现时间不准确的情况。

2．设置根用户密码

完成时区设置后单击 Next 按钮，安装程序将提示用户设置根用户（用户名为 root）密码，如图 1.28 所示。

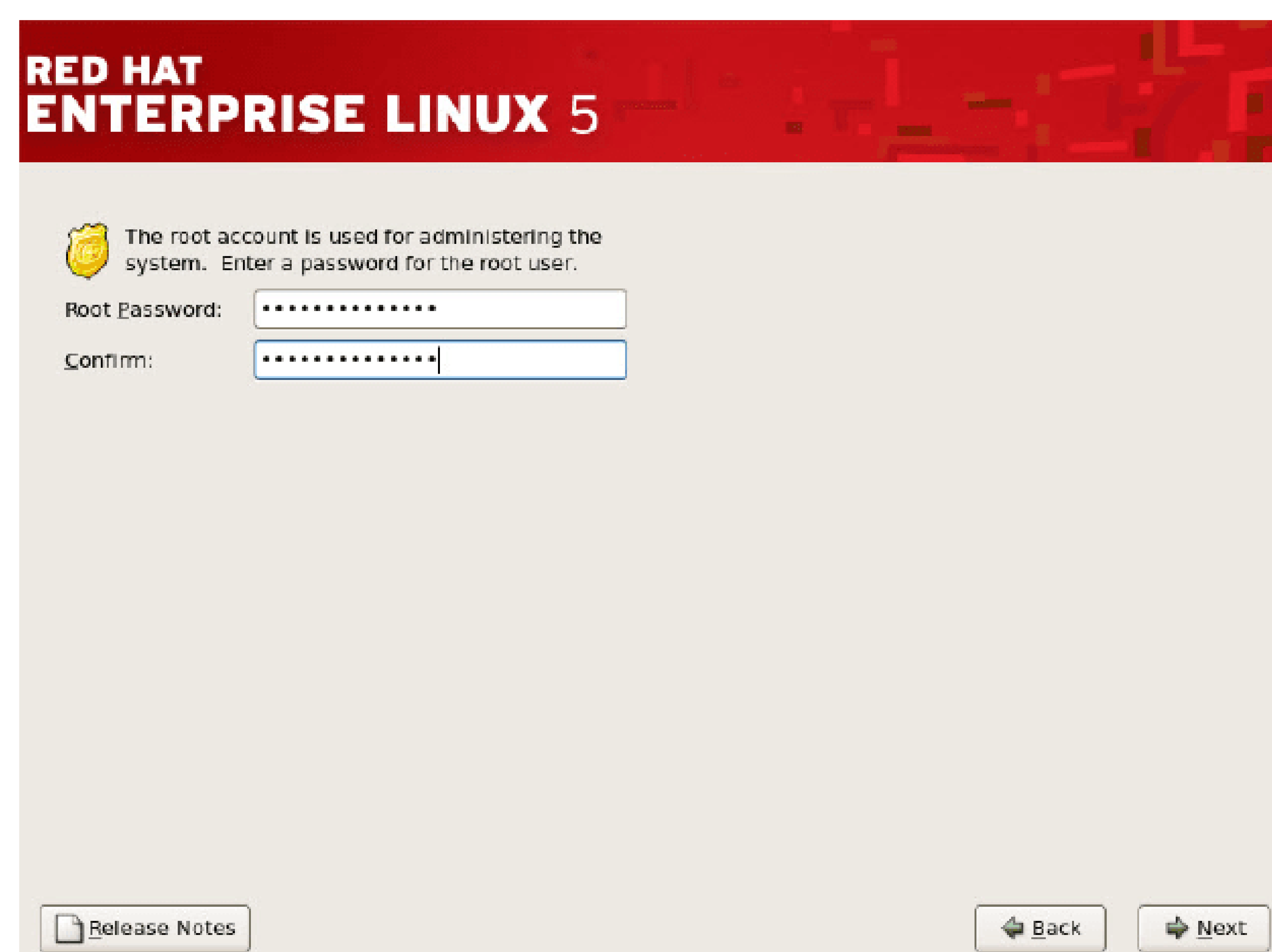


图 1.28　设置根用户密码

通常为了确保安全，设置密码需要注意两点：首先密码长度要足够长，通常建议 8～16 位；其次密码还应该具备一定的复杂性。

🔔小知识：在计算机安全中，判断一个密码的复杂性是否符合要求，通常使用“四分之三原则”衡量。“四分之三原则”是指密码中包含了可以构成密码的 4 种字符（大写字母、小写字母、数字和符号）中的 3 种。

根用户 root 在系统中拥有“至高无上”的权利（相当于 Windows 系统中的 Administrator 用户），因此应该为其设置一个足够复杂且容易记忆的密码。并且尽量不要使用一些容易被猜测到的字符串作为密码，例如电话号码、姓名的拼音、生日、电子邮件等，也不要将密码保存在容易泄露的地方。

1.4.8　定制软件包并开始安装

Linux 系统最大的特点是允许用户定制自己的操作系统，用户选择的软件包不同，安装完成后的操作系统也不同。为 root 用户设置密码后，安装程序将会提示用户定制随系统安装的软件列表，如图 1.29 所示。



图 1.29　定制安装的软件

（1）从图 1.29 中可以看出，软件包定制界面默认提供了两个选项：“Software Decelopment”（软件开发）和“Web server”（Web 服务）。用户可以直接勾选这两个选项，也可以勾选“Customize now”（现在定制）自定义安装的软件包。本例使用默认设置进行安装，单击 Next 按钮安装程序将检查软件包的依赖性。

（2）依赖性检查完成后，将提示用户即将安装 RHEL，并告知用户安装日志存放的位置等信息，如图 1.30 所示。

（3）确认后单击 Next 按钮即可开始安装过程，视安装软件包的数量和计算机配置不同，安装过程可能需要 20～50 分钟不等。

（4）安装完成后，系统会提示用户取出光盘等安装介质并重新启动系统，如图 1.31 所示。

（5）单击 Reboot 按钮安装程序会立即重新启动系统，到此安装过程就全部结束了。



图 1.30　RHEL5.3 即将开始安装



图 1.31　安装完成

1.4.9　第一次启动

第一次启动系统时，还需要做一些简单的设置才能进入系统。这些设置包括系统防火墙、日期时间和用户等。本小节将简单介绍这些设置。

1．防火墙设置

系统第一次启动时会首先要求用户接受其许可协议，接受之后将提示用户配置系统防火墙，如图 1.32 所示。



图 1.32　防火墙配置界面

提示：许多 Linux 发行版中都使用一个名为 iptables 的包过滤软件，作为系统默认的防火墙，iptables 功能非常强大，许多时候甚至可以顶替上万元的专业级防火墙产品。

默认将开启系统防火墙，并且允许 ssh 服务使用网络，ssh 服务主要用于远程访问和控制系统，通常推荐开启这个服务以便进行远程管理系统。

设置系统防火墙时，用户可以按实际情况在 Trusted services 列表中勾选允许使用网络的服务，也可以单击 Other ports 自定义使用网络的端口。如果不熟悉系统防火墙，此处可以使用默认设置。

注意：一个安全的防火墙策略可以极大地提升系统的安全性，关于防火墙的更多内容读者可以阅读更多专业书籍了解。

2．SELinux、Kdump、日期时间和软件更新设置

（1）配置完防火墙后，单击 Next 按钮，系统会提示用户设置 SELinux。SELinux 是一个系统安全的增强组件，可以大大提高 Linux 系统对未知攻击的防御能力。

提示：对于许多初学者而言，SELinux 可能并不实用，因此通常建议初学者关闭 SELinux。

（2）Kdump 是一个内核崩溃转存机制。当系统崩溃时。Kdump 可以将内存中的数据保存下来，以便于校正错误和灾难恢复。这个选项通常保持默认即可。

（3）第一次启动时，还要要求用户校正日期和时间，此步骤按需要进行操作即可。

（4）完成日期时间设置之后，系统会要求用户设置软件更新。此步操作需要用到红帽网络账号，如果没有，可以点选“No,I prefer to register at a later time.”以后再设置。

3．添加新用户及设置声卡

（1）许多 Linux 发行版在安装时都会提示用户创建一个新用户，RHEL 也不例外，如图 1.33 所示。

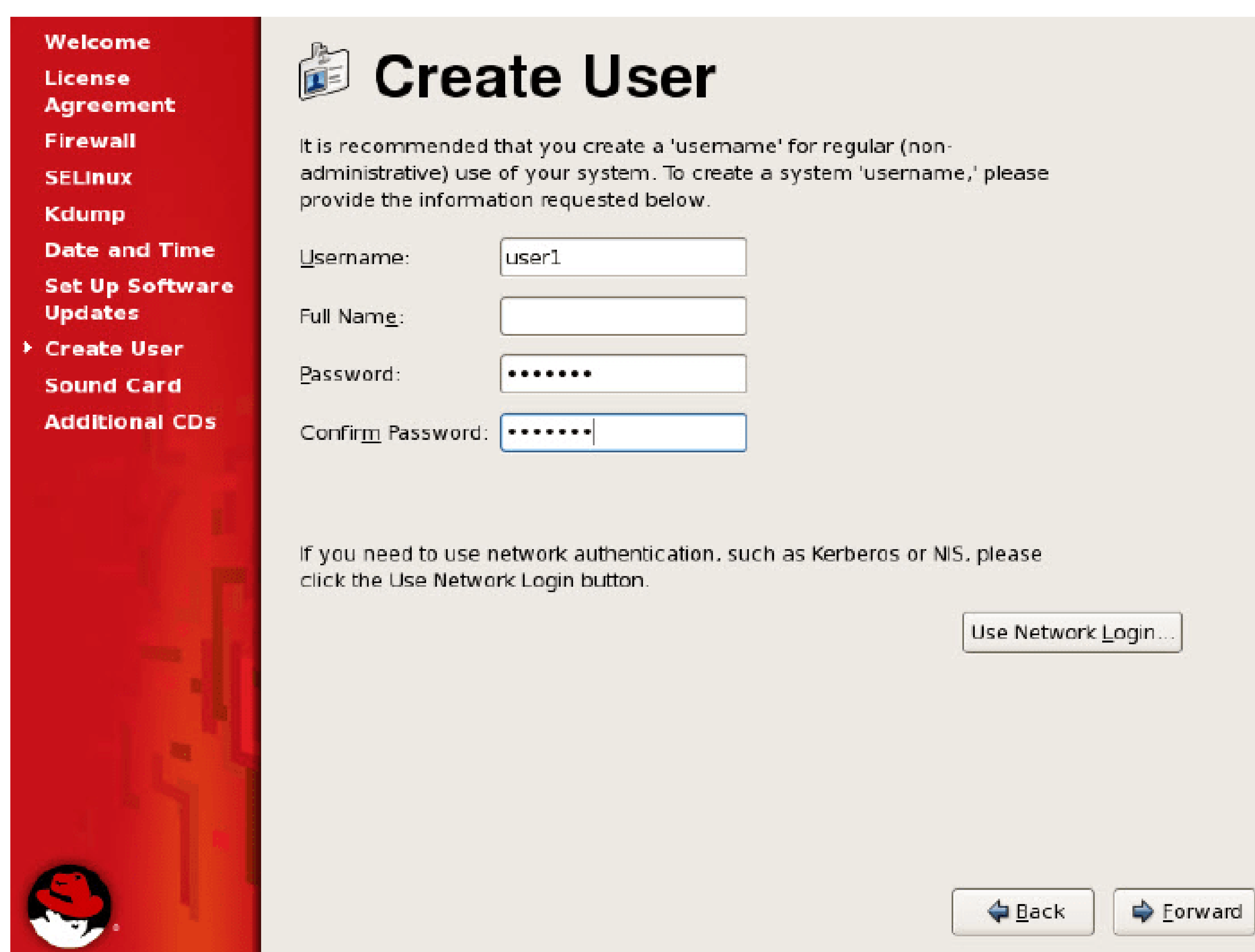


图 1.33 创建新用户

🔔技巧：Linux 系统安装完成后系统将会有一个默认用户 root，通常称其为根用户，根用户拥有系统的最高权限，可以对系统进行各种操作。因此通常建议使用普通用户登录系统，当需要执行一些管理命令时，再切换使用根用户，以免损坏系统中的数据。

在 Username 和 Full Name 文本框中输入用户的登录名及全名（全名可以为空），并重复输入两次密码之后，单击 Forward 按钮，此时系统会自动创建新用户。

（2）完成用户创建后，系统会要求用户设置并验证已安装的声卡，如图 1.34 所示。

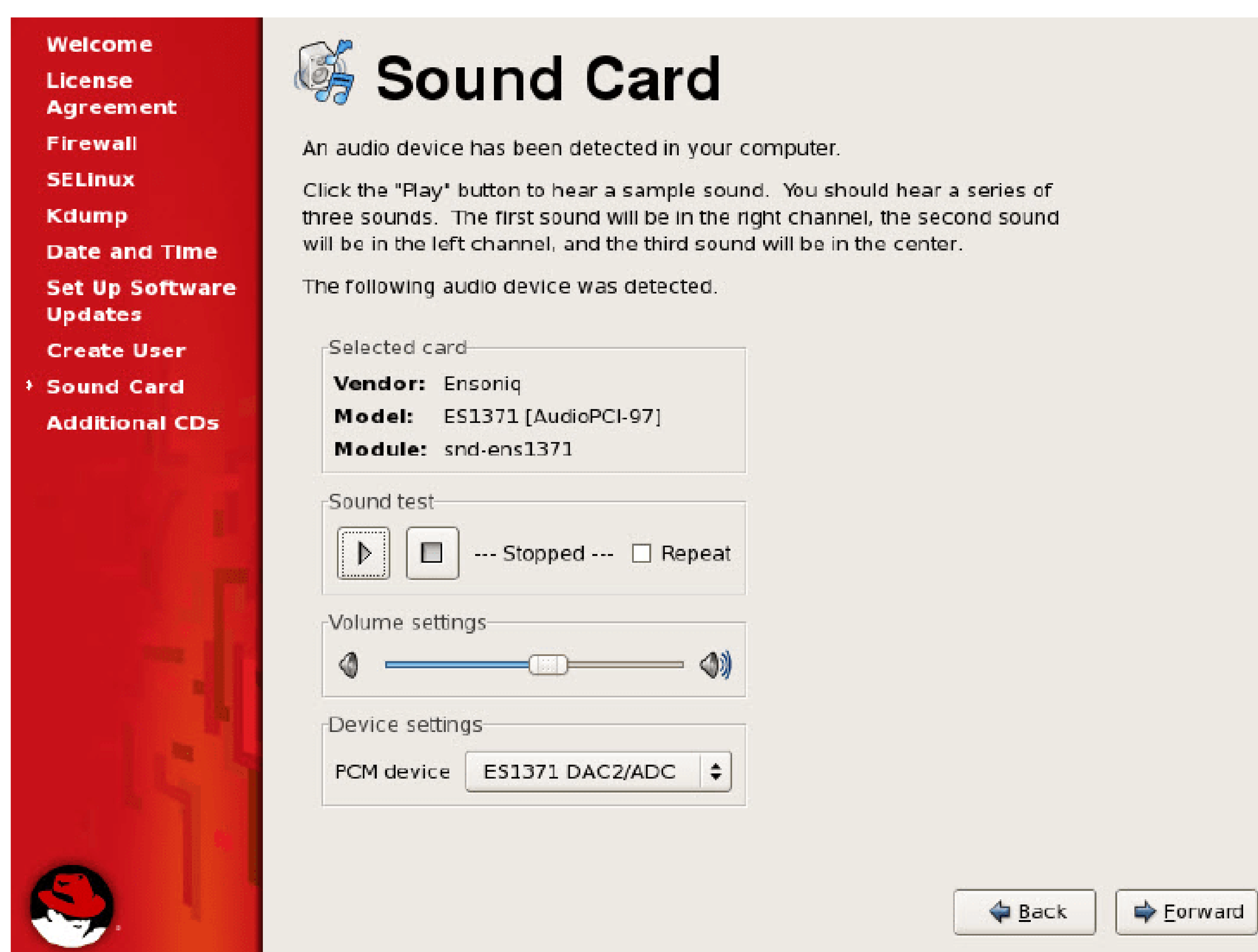


图 1.34 设置并验证声卡

测试声卡可以单击 Sound test 选项组中的播放按钮，系统会播放一段测试音乐，播放结束后按实际情况确认即可。

🔔注意：如果系统没有找到相应的声卡设备或声卡测试没有成功，可能需要检查声卡与主机的连接或安装相关驱动程序。

（3）声卡测试结束后，在 Additional CDs 界面中单击 Finish 按钮，即可完成 Linux 第一次启动的配置过程，此时用户已经拥有了一个可以使用的 Linux 系统。

1.5　小　　结

- 1.1 节介绍了 Linux 系统在不同环境中的应用、相对于其他系统的优势等内容。
- 1.2 节讲解了 Linux 系统中存储设备表示方法及目录结构。由于这部分内容与 Windows 系统差异较大，因此初学者应该特别注意。
- 1.3 节简单介绍了虚拟化及其在企业中的应用等内容。
- 1.4 节以 RHEL5.3 为例，讲解了 Linux 系统的安装过程，初步认识 Linux 的相关知识。

通过本章的学习，我们了解了 Linux 系统的基础知识，也迈出了走向 Linux 的第一步。在随后几个章节中，本书将介绍 Linux 系统的更多基础知识和使用技巧。

第 2 章　Linux 系统入门

本书第 1 章中以 RHEL5.3 为例，介绍了 Linux 操作系统的安装过程。至此我们迈出了走向 Linux 操作系统的第一步，接下来就可以登录并使用 Linux 系统了。

本章主要涉及的知识点如下。

- 简单介绍本书的知识结构、约定及学习的建议。
- 常见登录系统的方式及适用范围。
- 常见关闭、重启系统的命令及适用范围。
- 介绍如何获取 Linux 系统及命令帮助。
- 介绍 Linux 系统中的人机交互程序 Shell 及其分类，Bash Shell 的基本功能、管道及输入/输出的应用。

2.1　如何使用本书学习

长期以来，许多学习者都觉得学习 Linux 系统十分困难（特别是命令行模式的学习）。为更好地帮助初学者使用本书学习，本节将简单介绍本书的知识结构，以及学习 Linux 系统的建议。

2.1.1　本书的知识结构和约定

与市面上其他 Linux 书籍相比，本书更专注于介绍 Linux 系统管理，许多知识具备一定的专业性。因此在学习本书前，读者需要了解一些前提知识，本小节将简单介绍学习本书需要具备的前提知识及本书的知识结构。

1．前提知识

为更好地使用本书学习，建议初学者在学习本书前，应该了解一些计算机基础知识。按学习的目的不同，可能需要了解以下内容。

- 计算机硬件基础知识：了解计算机各种硬件及其功能、系统总线、计算机结构等。
- 计算机使用基础：包括计算机软、硬件及其简单使用。
- 存储设备基本知识：了解硬盘的基本原理、硬盘类型、使用硬盘存储数据的方法等。
- 计算机网络基础知识：包括 IP 地址分类、第二、三层网络寻址方法和 VLAN 等内容。

2．本书的知识结构

本书的主要知识结构如图 2.1 所示。本书讲解 Linux 基础知识的同时，也包含了一些

高级应用和排除错误等知识，因此使用本书学习的过程中，读者需要注意自主思考和验证。与其他同类书籍相比，本书更加注重实例和应用，书中列举了大量 Linux 系统管理和 Shell 脚本实例，这些实例都可以直接应用于实际环境。

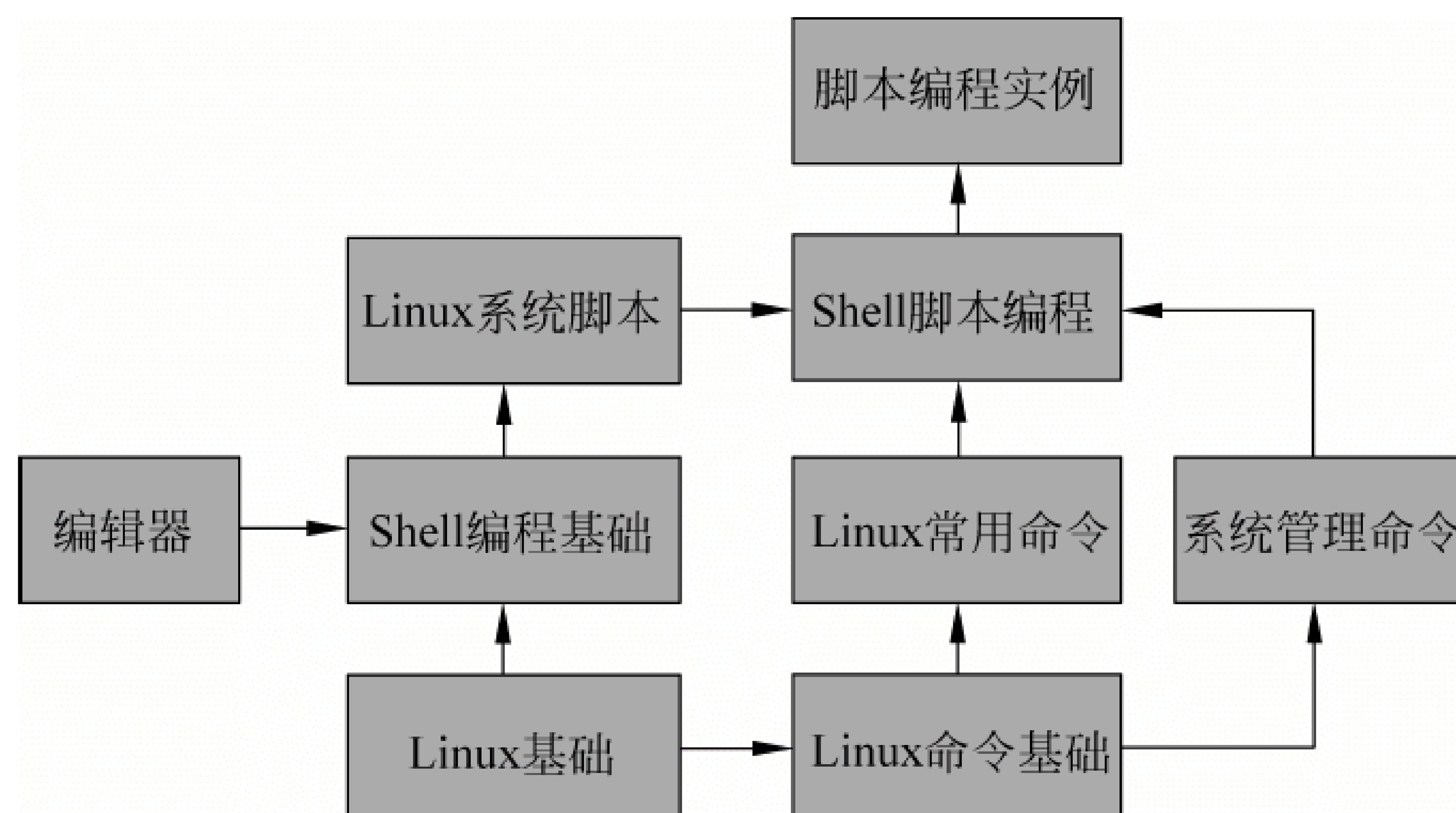


图 2.1　本书的知识结构图

3．约定

由于本书中的知识面较广且前后交错，为帮助读者更好地学习，在书中使用了以下批注。

- 注意：操作可能包含危险、需要谨慎操作、相关知识链接、本书内容的声明等。
- 提示：可能需要额外执行的操作、容易忽略的前提操作、容易忽略的问题等。
- 小知识：需要引起注意的知识、可以扩展阅读的相关内容等。
- 技巧：实际应用中经常使用的技巧、解决问题的思路等。
- 说明：关于书中描述内容的声明、与书中描述不相符的其他情况等。
- 图片：由于印刷方面的要求，本书中的所有字符界面的图示均采用图形界面中的终端的形式呈现（内容和操作都与字符界面相同）。

为了更好地讲解命令，本书中大部分命令用法示例中，用户输入的命令都使用加粗的方式表示，命令注释都使用“#”作为开头。为方便读者阅读，本书采用了中文注释，但在实际操作时，包括注释在内的任何内容都不建议使用中文（使用中文可能会引发一些问题）。

4．关于图形界面

目前许多发行版都采用仿 Windows 系统的图形界面，虽然使用仿真的图形界面也可以管理 Linux 系统，但本书主要采用字符界面和命令行模式进行讲解，主要原因如下。

- 虽然可以使用图形界面管理 Linux 系统，但目前图形界面仍然有许多局限性，不能完成所有 Linux 系统管理任务（例如不能编译安装软件）。
- 与图形界面相比，命令行模式更高效便捷。
- 由于使用字符界面可以节省更多系统资源，因此目前许多运行中的 Linux 服务器系统都使用字符界面。
- 系统发生错误时可能无法进入图形界面。

基于以上这些原因，本书大多数情况下都在命令行模式或命令终端中进行讲解，图形界面留给读者自行研究。

2.1.2　学习 Linux 系统的建议

许多初学者在学习 Linux 系统的过程中会遇到各种困难，为帮助初学者更好地学习，本小节为初学者提供了学习 Linux 系统的几个建议。

（1）关于命令

许多初学者都对记忆大量命令望而生畏（这也是 Linux 系统难点所在），但经验丰富的用户却并不认为使用 Linux 系统必须记忆大量命令。经过统计，完成常见的 Linux 系统管理任务，使用的命令及常见用法的总量仅有 100 余个。并且这些命令和常见用法在使用和管理 Linux 系统的过程中，需要经常重复使用。由此可见，只要不断思考和练习，学会并熟练使用命令，并不是一件十分困难的事。

通常建议初学者从最基本的系统帮助出发，学习使用 Linux 系统的方法。遇到问题多使用系统提供的帮助，而不是死记硬背命令和用法。

（2）关于笔记

通常建议初学者多练习一些常用的命令，而将一些不常用的命令及其用法备注，写入博客、日常笔记等容易查看的文档或笔记中，以便于需要时可以快速查看。

（3）关于命令行模式

大多数人使用命令行模式都觉得不方便，例如无法从互联网上查阅相关帮助。这时可以通过远程或图形界面的方式登录到 Linux 系统中，一边阅读说明一边验证学习，待熟悉之后再使用命令行模式。

2.2　登 录 系 统

要使用 Linux 系统，首先需要启动并登录系统，与 UNIX 类似，Linux 也支持多种登录方式。按登录系统的位置可以将登录方式分为如下两种。

- ❑ 本地登录：通常用于笔记本等桌面系统或服务器出现故障无法连接网络的情况。本地登录可以使用图形界面和命令行模式（有些书籍也称为字符界面）两种方式。
- ❑ 远程登录：大多数多用户系统和服务器都使用远程登录。远程登录可以使用 SSH、Telnet、VNC、SFTP 4 种方式。

不同的登录方式适用于不同的环境，本节将简单介绍登录系统的常见方式及适用的环境。

2.2.1　图形界面登录

虽然图形界面一般用于桌面环境，但我们也建议初学者使用图形界面及其中的部分图形化工具学习，例如使用图形化的 Web 浏览器一边查找帮助一边学习等。本小节将简单介绍如何在图形界面中登录系统。

在 RHEL5.3 中，第一次登录系统默认使用图形界面的方式进行登录，如图 2.2 所示。

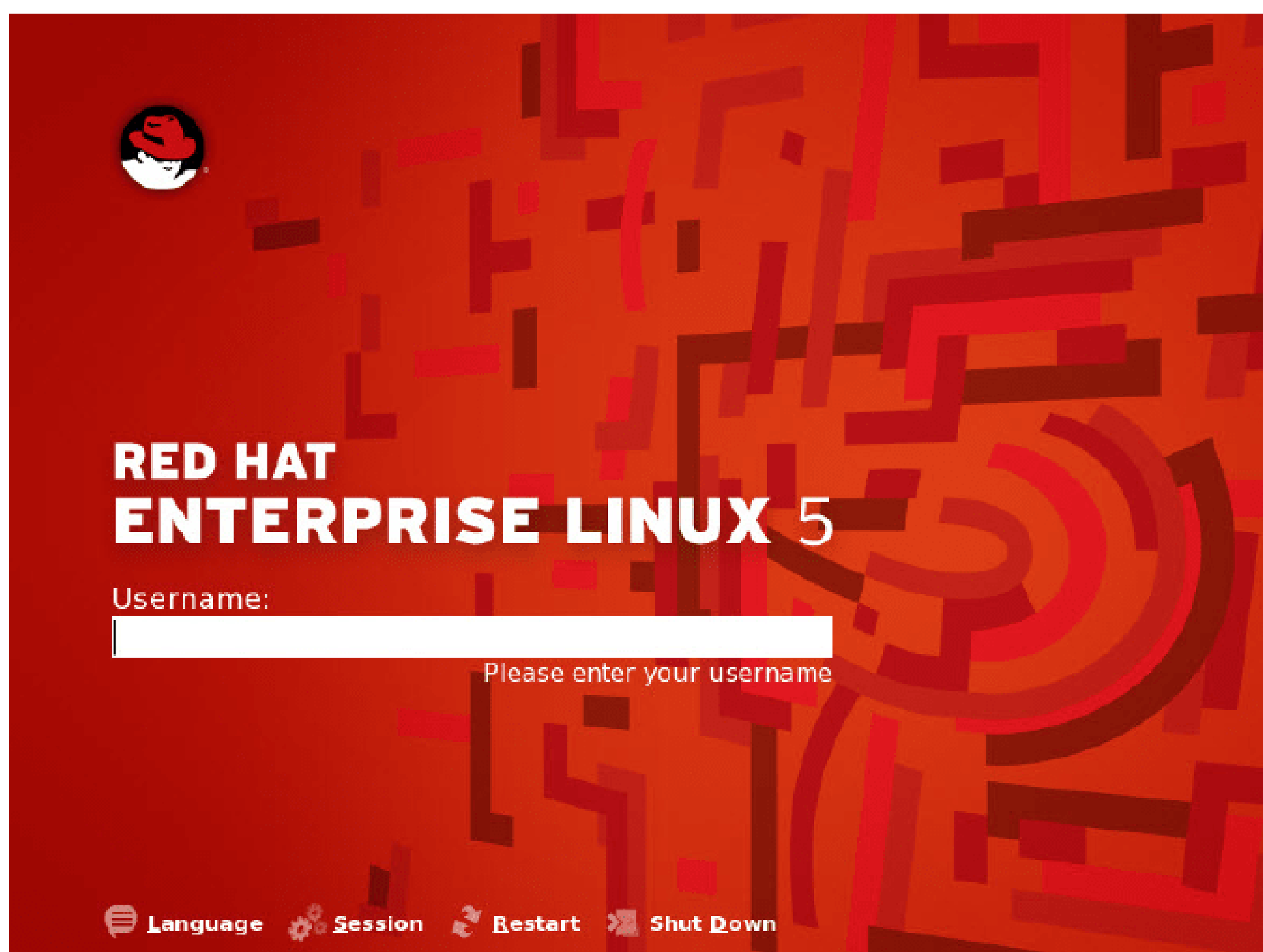


图 2.2　图形登录界面

在图形登录界面的下方提供了一些简单的选项，这些选项如下。

- ❑ Language 选项：可以选择系统使用的语言（前提是安装系统时已经安装了相关的语言包）。
- ❑ Session 选项：用于选择会话的类型。
- ❑ Restart 和 Shut Down 选项：可以重启和关闭系统。

以图形界面进行登录时，直接输入用户名和密码即可登录系统。如果使用 Language 选项更改了系统语言，此时将会提示是否要将更改后的语言设置为系统默认语言，按需要进行选择之后即可进入 Linux 的图形界面。

提示： 虽然 Linux 图形界面提供了中文界面，但大多数发行版的内核都未添加中文支持，因此任何与系统有关的配置文件中都不要使用中文(这可能会导致一些意外的问题)。

RHEL5.3 的图形界面如图 2.3 所示，这是一个使用 GNOME 的图形界面，图形模式下自带的工具几乎都可以在图形界面左上角的菜单中找到。

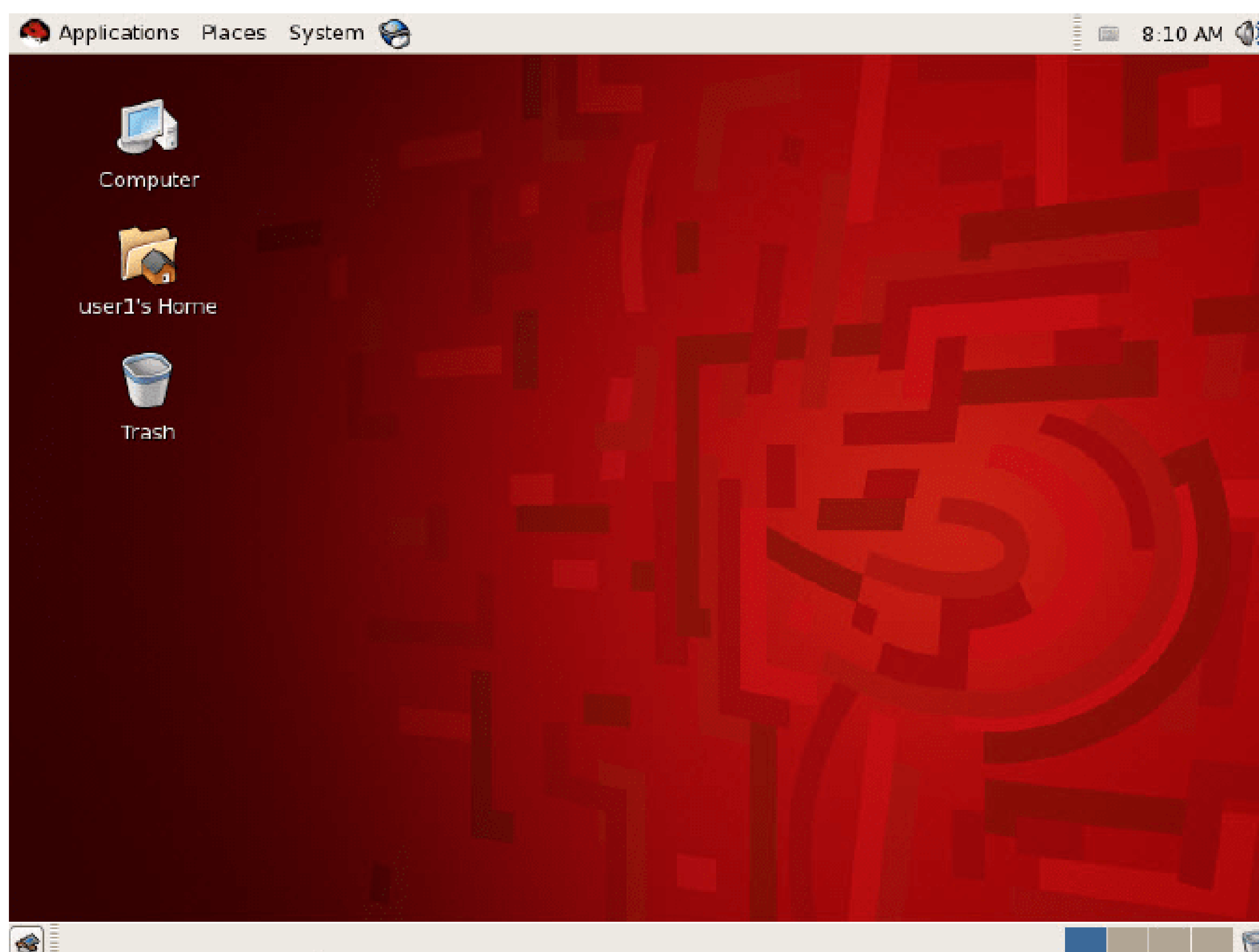


图 2.3　RHEL5.3 的图形界面

对于不熟悉 Linux 的初学者而言，使用图形界面有以下优势：

- 在图形界面终端中，能够较好地支持命令中的多种语言。
- 初学者可以通过图形化的浏览器从互联网上获得更多的帮助。
- 使用图形环境中的工具能够更好地学习 Shell 编程（这些工具将在编辑器相关章节中介绍）。

图形界面还有许多优势（例如可以查看多语言帮助等），此处不一一列举，感兴趣的读者可以阅读相关文档了解其使用方法。

2.2.2　命令行登录

命令行模式（也称字符模式或字符界面）是 Linux 系统的精华部分，其高效、快捷、方便的特点是图形界面和 Windows 操作系统不能比拟的。命令行模式是每一个 Linux 学习者都需要学习的部分，其地位非常重要（原因很多，例如排除错误时可能只能使用命令行模式等）。本小节将简单介绍如何在命令行模式中登录。

提示： 命令行模式与微软的 DOS 操作系统界面有天壤之别，读者可以阅读相关文档了解此部分内容。

（1）命令行登录系统

默认情况下，RHEL5.3 启动后会进入图形界面，如果需要从图形界面切换至命令行模式，可以先使用 root 用户登录系统，然后右击桌面，并在右键菜单中单击 Open Terminal 打开终端，在终端窗口中执行命令 init 3，按 Enter 键即可进入命令行登录界面。

与图形界面不同的是，命令行登录界面仅包含当前操作系统的版本信息等内容，如图 2.4 所示。

```
Red Hat Enterprise Linux Server release 5.3 (Tikanga)
Kernel 2.6.18-128.el5 on an i686

localhost login: _
```

图 2.4　命令行登录界面

在图 2.4 中可以看到，系统使用主机名加上字符“login:”作为登录的提示字符，此时只需要输入用户名和密码（输入密码时，系统出于安全考虑，用户输入的密码都不会被显示出来）即可登录系统。登录成功后，用户就可以看到命令行模式的提示符（即命令提示符）。当输入的用户名或密码不符时，系统将会提示用户 Login incorrect，并重新返回登录提示符。

用户成功登录后，系统会自动将用户的工作目录定位到用户的家目录（有时也称为用户的主目录）中。家目录是用户登录系统后的起始目录，root 用户的家目录位于/root，普通用户的家目录通常位于/home/*username*（其中 *username* 为用户名）。家目录通常用于存放系统、应用程序的初始化文件，以及用户的个人文件等。

小知识： 终端是与主机系统相连的用户设备，作用是将用户的输入交给主机系统，并将主机系统的处理结果通过终端显示给用户（可以将其理解为用户与主机系统交互的接口）。Linux 系统中存在多种类型的终端，包括串行终端/dev/ttyS*n*、控制终端/dev/tty、控制台终端/dev/tty*n* 和/dev/console、虚拟终端/dev/pts/*n* 等（*n* 为终端号）。本书中的终端大多是指由软件生成的虚拟终端。

（2）命令提示符

用户在命令行模式中成功登录后，如果使用 root 用户登录将会返回命令提示符[root@localhost ~]# ，使用普通用户登录则返回提示符[user1@localhost ~]$ 。这个提示符可以拆解为几部分，这几部分的含义如下。

- root 和 user1：表示当前登录系统的用户名，通过此部分可以确定当前登录系统的用户名。
- 用户名后面是使用符号“@”（与英语单词 at 读法相同）相隔的主机名，在上面两个例子中主机名均为 localhost。
- 主机名后面是当前用户所在的目录。“~”表示当前用户所在的目录为用户的家目录。
- 在提示符的最后是“#”和“$”，“#”表示当前登录的用户是根用户，“$”表示当前登录系统的是普通用户。使用此符号可以判断当前登录系统的是否为根用户。

命令提示符的作用是告诉用户已经成功登录系统，可以在“$”或“#”符号后面输入要执行的命令。

由于使用了不同的环境变量，因此不同系统使用的命令提示符有所不同，关于命令提示符的相关变量，读者可以参考本书第 15 章的相关内容。

2.2.3　SSH 远程登录

如果初学者觉得在虚拟机中学习 Linux 系统很不方便（可能的麻烦是每次都要使用组合键 Ctrl+Alt 切换控制目标），这时建议使用远程方式登录系统，一边查询一边学习。本小节将简单介绍远程登录的方式和 SSH 远程登录。

1．远程登录的方式

常见的远程登录方式有 SSH、Telnet、VNC 和 SFTP 4 种。

- SSH：SSH（Secure Shell）是一种使用加密技术保护传输数据包的远程登录工具，所有数据包都先经过加密，再进行传输。由于 SSH 是一种安全性非常高的远程登录工具，因此 SSH 也是 Linux 系统中使用最广泛的远程登录方式。
- Telnet：Telnet 是一个传统的交互式登录工具。与 SSH 不同的是，Telnet 并没有使用加密技术，所有内容都通过明文方式传输。由于其保密性差，因此通常将其应用到能够确认安全的环境下，例如一些私有网络等。
- VNC：VNC（Virtual Network Computing，虚拟远程计算机）是由 AT&T 欧洲实验室开发的一个用于远程控制的开源软件，在 Linux 系统中主要用于远程桌面控制。
- SFTP：SFTP（Secure File Transfer Protocol，安全文件传输协议）是 SSH 的一部分，主要用来在 Linux 系统间传送文件。

由于大多数发行版都默认安装了 SSH 且其安全性较高，因此通过 SSH 远程登录并管理 Linux 系统已经成为管理员的必备技能之一。这也是推荐初学者通过 SSH 远程登录学习命令行模式的重要原因，本小节将介绍如何在 Windows 和 Linux 系统中使用 SSH 远程登录。

2．Windows系统中的SSH登录

在 Windows 环境中使用 SSH 登录需要借助一些工具，常用的工具有 PuTTY 和

SecureCRT 等，运行界面分别如图 2.5 和图 2.6 所示。

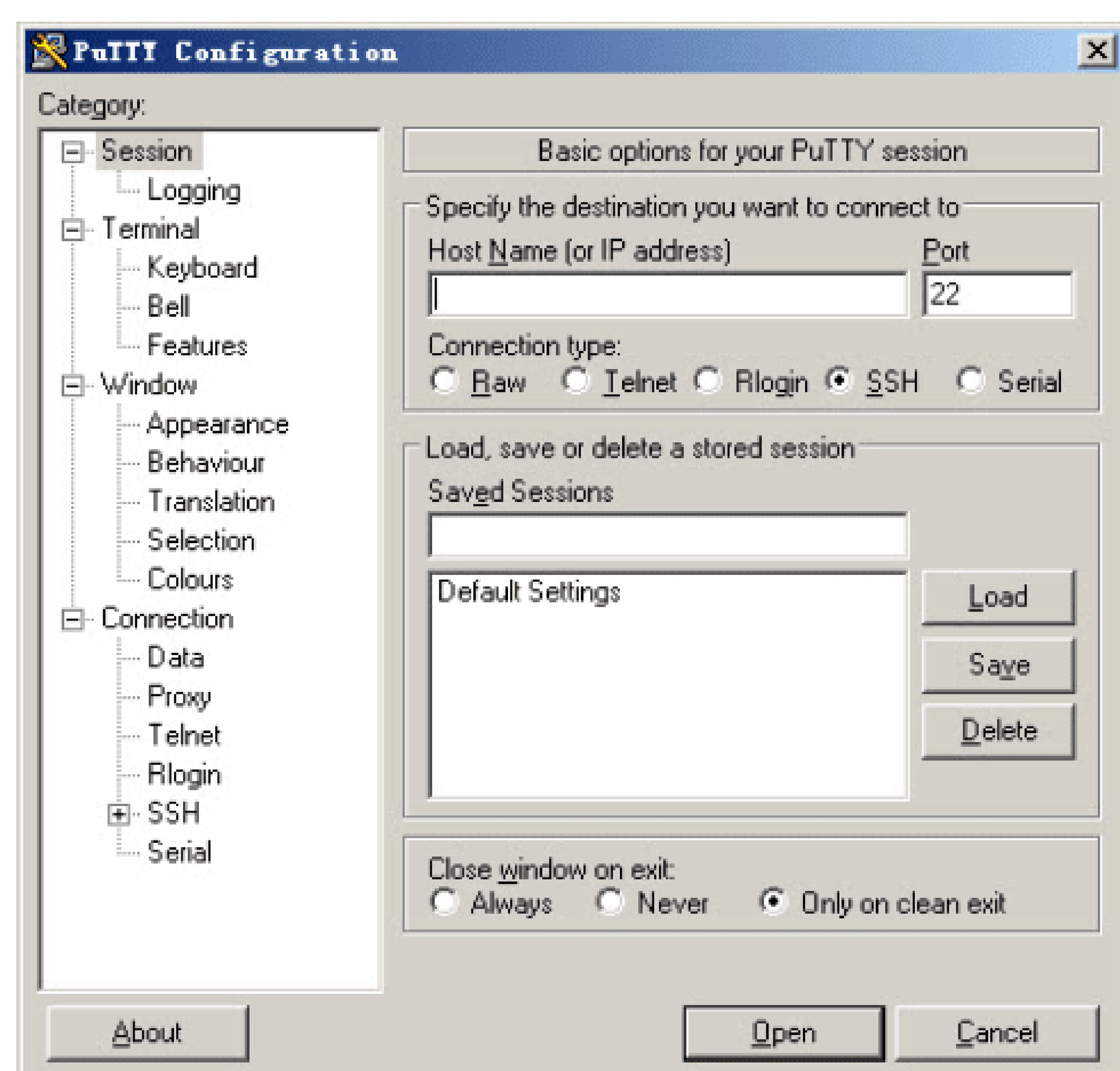


图 2.5　PuTTY 运行界面

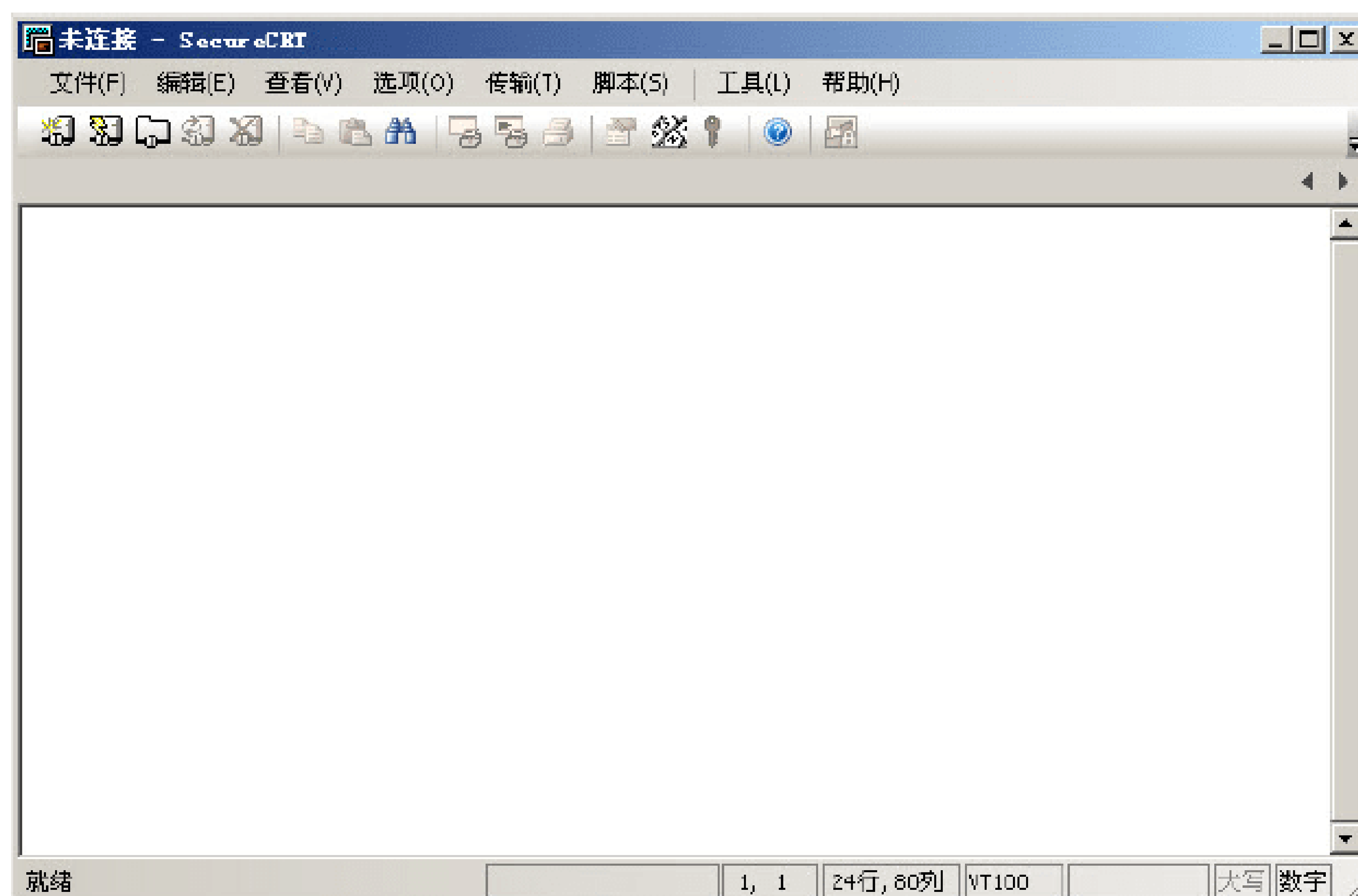


图 2.6　SecureCRT 运行界面

（1）PuTTY 是登录 Linux 系统最简便的工具之一，拥有体积小巧、操作简单、功能强大等特点，许多 Linux 管理员都使用 PuTTY 作为远程登录的工具。

在 PuTTY 主界面中输入远程 Linux 系统的 IP 地址和相应的端口号，单击 Open 按钮即可使用 SSH 进行登录。

（2）如果需要远程登录的 Linux 系统较多，通常推荐使用 SecureCRT 作为远程登录的工具。SecureCRT 不仅可以保存多个远程登录连接，还可以将多个登录终端放置在当前窗口的标签页中，非常适合需要登录多个 Linux 系统的情况。

如果要使用 SecureCRT 工具登录 Linux 系统，可以在其主界面中单击“新建连接”按钮，然后在新建连接向导中按提示操作即可新建并保存连接。

3．Linux系统中的SSH登录

在 Linux 系统中要使用 SSH 远程登录到另一个 Linux 系统，可以使用 ssh 命令加用户名和 IP 地址的方法。例如：

```
[root@localhost ~]# ssh root@192.168.44.**
The authenticity of host '192.168.44.** (192.168.44.**)' can't be established.
#提示用户 RSA 密钥指纹
RSA key fingerprint is 8f:9a:51:93:e5:c3:03:97:63:b5:53:bd:e7:f1:6b:5c.
#询问用户是否需要继续连接，以下粗体部分为用户的输入
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.44.**' (RSA) to the list of known hosts.
#下面需要输入 root 用户的登录密码
root@192.168.44.**'s password:
Last login: Sat Sep  4 21:49:56 2010 from 192.168.44.***
#使用 exit 命令断开远程登录
[root@localhost ~]# exit
Connection to 192.168.44.** closed.
```

```
[root@localhost ~]#
```

SSH 可以使用用户密码和 Public-Key Infrastructure（PKI）两种认证方式登录。本小节中仅以用户密码认证为例介绍 SSH 登录，读者可以自行阅读相关文档了解和使用 PKI 认证。

提示：在 RHEL5.3 中可以使用命令“service sshd start”（其他操作系统中可以使用“/etc/init.d/sshd start”）启动 SSH 服务；如果系统防火墙阻挡用户访问，可以使用命令“service iptables stop”（其他系统中可以使用“/etc/init.d/iptables stop”）关闭系统防火墙。

2.2.4　Telnet 远程登录

虽然 Telnet 的安全性较差，但却具备更广泛的应用环境，例如用户可以从路由器、交换机、Windows 等设备和系统中使用 Telnet 进行远程登录。本小节将简单介绍 Telnet 远程登录系统。

提示：目前许多 Linux 发行版默认都不会安装 Telnet，因此在使用 Telnet 远程登录前，还需要添加 Telnet 服务，并设置防火墙规则。

在 Linux 系统中使用 Telnet 远程登录系统时，可以使用 telnet 命令：

```
[root@localhost ~]# telnet 192.168.56.131
Trying 192.168.56.131...
Connected to 192.168.56.131 (192.168.56.131).
Escape character is '^]'.
Red Hat Enterprise Linux Server release 5.3 (Tikanga)
Kernel 2.6.18-128.el5 on an i686
#Telnet 登录要求用户输入用户名和密码
login: user1
Password:
Last login: Fri May  6 03:58:24 from 192.168.56.129
[user1@localhost ~]$
[user1@localhost ~]$ exit
Connection closed by foreign host.
[root@localhost ~]#
```

由于在 Windows 系统中使用 Telnet 远程登录与 Linux 系统类似，因此本书中仅以 Linux 系统为例介绍如何使用 Telnet 远程登录。其他系统中的 Telnet 远程登录，读者可以参考相关文档。

2.2.5　VNC 远程登录

如果初学者不习惯在虚拟机中使用 Linux 图形界面，可以使用 VNC 远程图形界面的方式学习（远程图形界面类似于 Windows 系统中的远程桌面）。本小节将介绍如何使用

VNC 软件以图形化界面远程登录。

VNC 远程登录需要使用一个名为 VNC Viewer 的软件，该软件拥有 Windows 和 Linux 两个版本（Linux 系统中的 VNC Viewer 需要在图形界面中使用）。Windows 系统中的 VNC Viewer 运行界面如图 2.7 所示。

使用 VNC 远程登录时，在 VNC Viewer 软件中输入远程主机的 IP 地址和桌面号，然后单击 Connect 按钮执行连接（图 2.7 中的 172.16.1.232:1 表示主机 172.16.1.232 的 1 号桌面）。连接后会提示用户输入相应桌面的密码（此处应该输入桌面密码而非系统用户密码），输入后即可成功登录。VNC 远程登录的图形界面与 Linux 系统的图形界面完全相同，如图 2.8 所示。

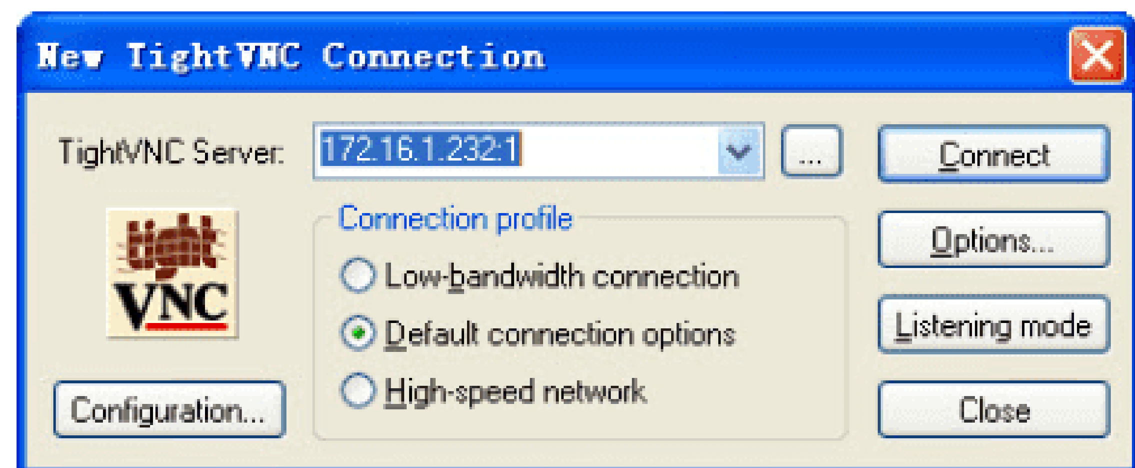


图 2.7　VNC 运行界面

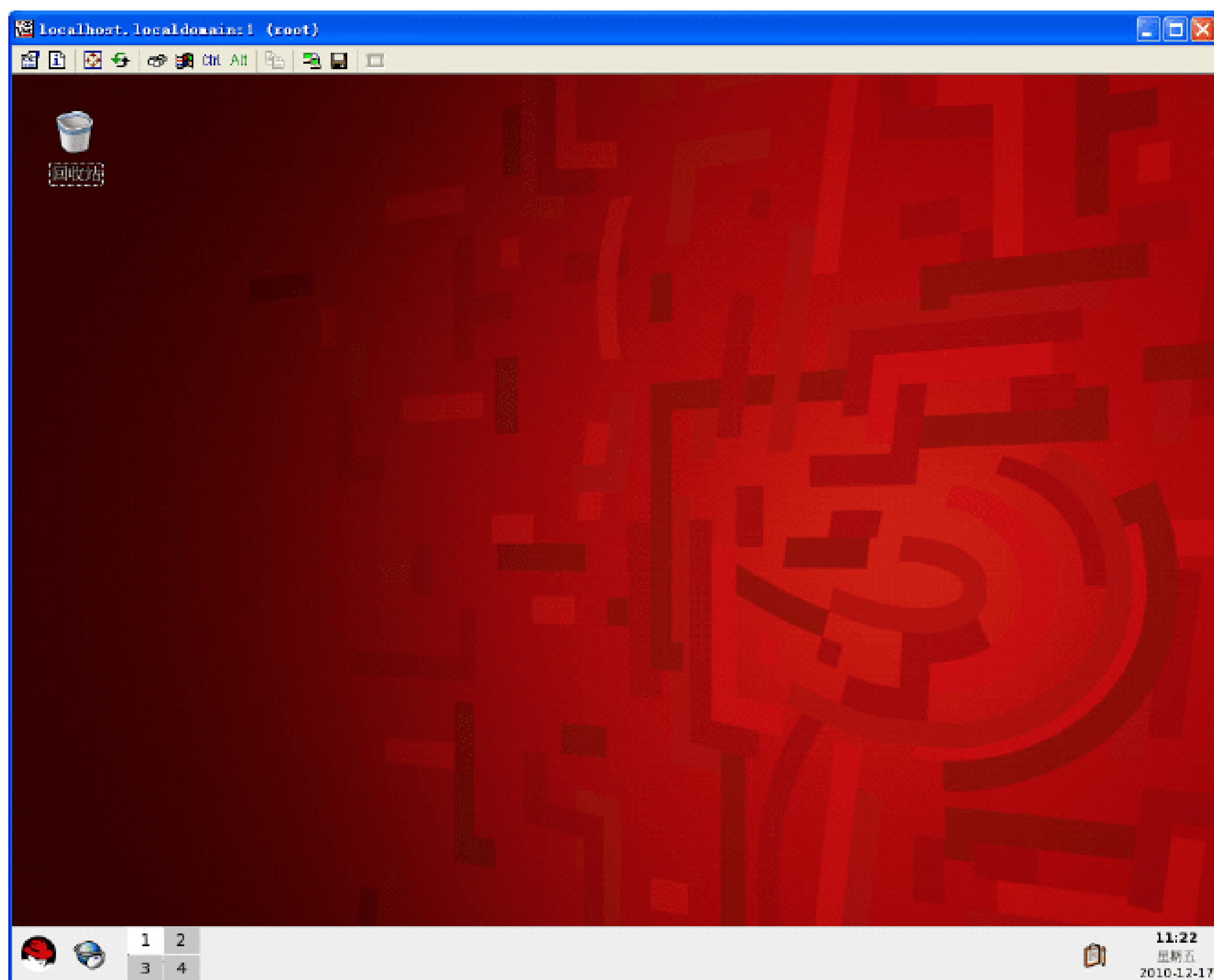


图 2.8　VNC 远程登录界面

提示：如果需要在 RHEL5.3 中设置并开启 VNC 桌面，可以使用命令 vncserver :1（:1 表示 1 号桌面），并设置密码即可启用第 1 个桌面。但这时的桌面仅是一个终端，如果需要使用图形界面，可以阅读相关文档了解更多的配置方法。

2.2.6　SFTP 登录

管理员通常都使用远程的方式管理服务器，有时可能需要向服务器传送文件，这些文件可能是一些小的脚本文件，或从互联网上下载的相关程序等。传送文件可以有许多方法，例如 FTP、Samba 文件服务器、NFS 网络文件系统等，其中最简便的方法是使用 SFTP 登录并传送文件。本小节将介绍如何使用 SFTP 登录并向远程主机传送文件。

1．Windows系统中的SFTP登录

Windows 系统中可以使用的 SFTP 登录工具有 WinSCP 和 SecureFX，这两个工具的主要作用是从 Windows 主机远程登录到 Linux 服务器，并在二者之间进行文件传送。

此处以 WinSCP 工具为例简单讲解如何传送文件。首先启动 WinSCP 工具，如图 2.9 所示。在 WinSCP 登录界面中输入主机 IP 地址、用户名、密码后就可以使用 SFTP 登录了，登录后的界面如图 2.10 所示。从 WinSCP 工作界面中可以看出，WinSCP 与 FTP 软件界面十分类似，因此可以像操作 FTP 软件那样使用 WinSCP 传送文件。

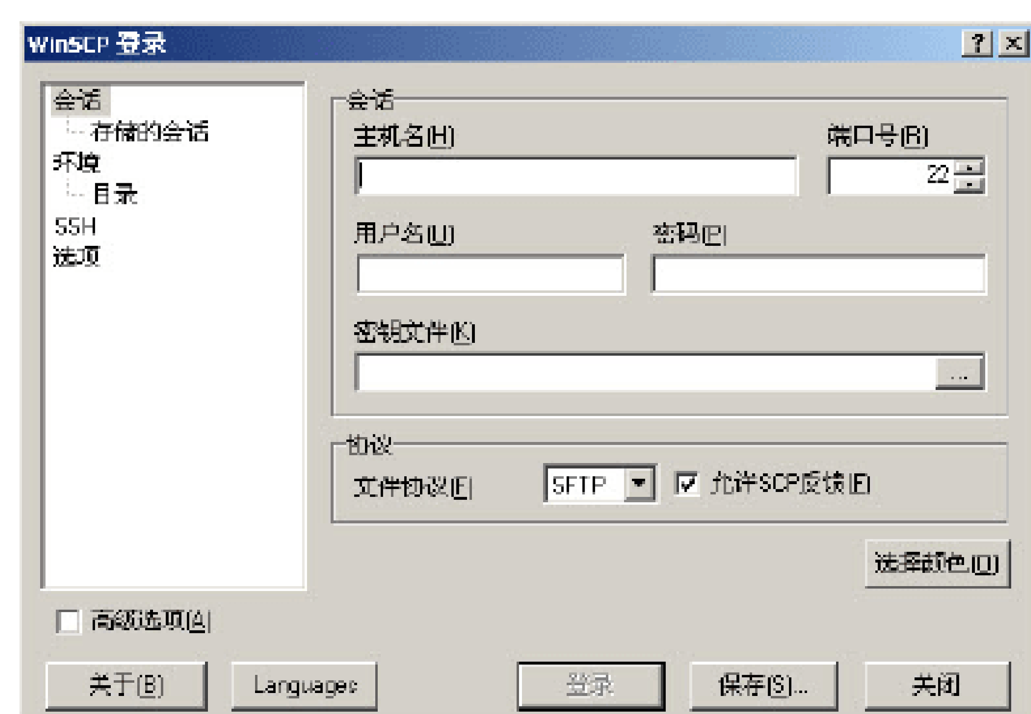


图 2.9　WinSCP 登录界面

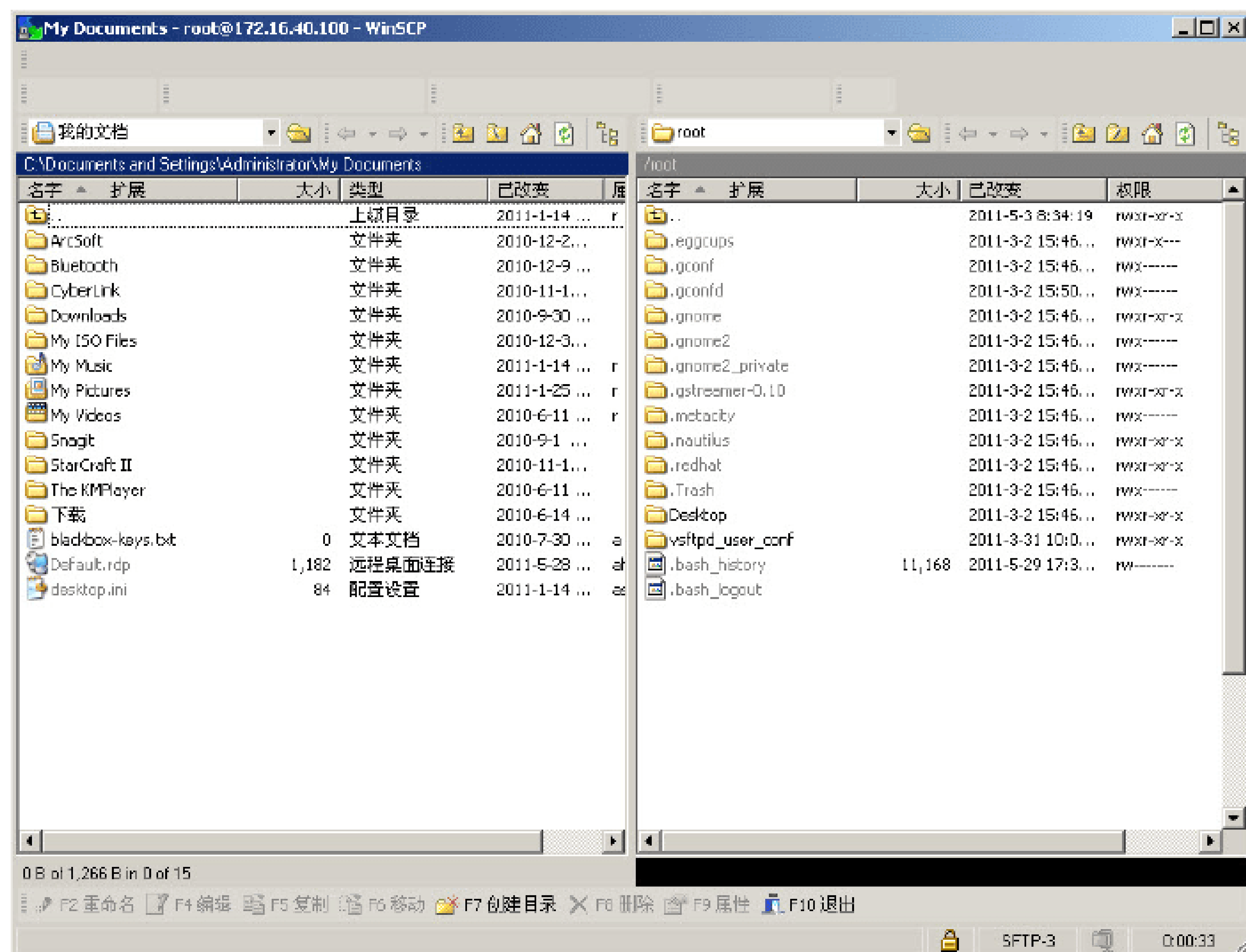


图 2.10　WinSCP 工作界面

2．Linux系统中的SFTP登录

有时需要在 Linux 系统间传送文件，在 Linux 系统中使用 SFTP 时，可以使用命令 sftp：

```
#以 root 用户身份登录远程服务器
# sftp root@192.168.44.102
root@192.168.44.102's password:
Connected to 192.168.44.102.
#使用 get 命令下载远程服务器的文件/root/a
sftp> get /root/a
Fetching /root/a to a
/root/a                          100% 9733    9.5KB/s   9.5KB/s   00:00
#使用 put 命令上传本地文件/root/test1
sftp> put /root/test1
Uploading /root/test1 to /root/test1
/root/test1                      100% 9733    9.5KB/s   9.5KB/s   00:00
#使用 bye 命令退出 SFTP 远程登录
sftp> bye
```

sftp 命令与 Linux 系统中的 ftp 命令十分类似，读者可以参考 ftp 命令使用 sftp。

注意：由于本书中的大多数命令都不涉及命令提示符，因此大多数命令示例中都使用了命令提示符简写。使用#开头的命令为 root 用户执行的命令，$开头的命令为普通用户执行的命令。

2.3　关闭、重启系统

Linux 是一个多用户、多任务系统，如果不正确地关闭或重启系统，可能会导致系统中的用户数据丢失。可能的情况是用户正在执行某个关键的运算或操作等，如果不经提示关闭系统，用户将来不及保存当前数据，从而导致数据丢失。因此应该掌握正确地关闭、重启系统的方法，以避免数据丢失。本节将介绍关闭、重启系统常用的命令。

2.3.1　关闭系统命令之 shutdown

shutdown 命令是最常用的关闭系统命令，不仅可以用于立即关闭系统，还可以在指定时间关闭系统。本小节将简单介绍如何使用 shutdown 命令关闭、重启系统。

（1）在 1 分钟后关闭系统：

```
# shutdown -h 1
```

由于关闭、重启系统需要管理员权限，因此应该以 root 用户的身份执行 shutdown 等关闭、重启系统命令。

立即关闭系统：

```
# shutdown -h now
```

（2）指定时间关闭系统：

```
# shutdown -h 15:30
```

当根用户发出关机指令后，系统中的其他用户可以得到如下提示：

```
The system is going DOWN for system halt in 2 minutes!
```

如果得到此提示信息，用户应该立即停止正在进行的作业，保存必要的数据然后退出系统。

（3）shutdown 命令也可用于重启系统：

```
# shutdown -r 2
```

系统将在 2 分钟后重新启动。

如果正在使用系统的用户不止一个，关闭系统时应该使用 shutdown 命令，并采用延时关闭的方法，以避免数据丢失。

2.3.2　关闭系统命令之 poweroff

如果确认系统中已经没有用户存在且所有数据都已保存，需要立即关闭系统，可以使用 poweroff 命令。

使用 poweroff 立即关闭系统：

```
# poweroff
```

2.3.3　挂起系统命令 halt

halt 命令用于挂起系统（挂起系统通常应用于笔记本等便携设备中），与 shutdown 和 poweroff 命令不同，使用 halt 只会挂起系统（系统停止运行）而不会切断主机电源。

使用 halt 立即挂起系统：

```
# halt
```

注意：由于挂起系统需要多种硬件驱动支持，因此使用挂起前应该在系统中作相应的设置，否则可能会出现无法恢复系统的现象。

2.3.4　重启系统命令 reboot

重启系统时，如果确定系统中已经没有任何用户的数据需要保存，可以使用 reboot 命令。

使用 reboot 命令重启系统：

```
# reboot
```

使用了 reboot 命令之后，系统将会立即关闭并重启。

2.3.5　切换系统运行级别命令 init

init 命令用于改变系统的运行级别，改变系统的运行级别也可以关闭或重启系统。本小节将介绍如何使用 init 命令切换系统的运行级别。

（1）运行级别

运行级别在 Linux 系统中非常重要，它描述了 Linux 系统的运行状态，也是 Linux 系统管理知识中不可或缺的部分。Linux 系统使用数字 0～6 来表示系统运行的状态（即运行级别），这 7 个运行级别分别如下所示。

- 0：数字 0 表示停机，当运行级别切换至 0 时，系统会立即关闭正在运行的服务，并关闭系统电源。
- 1：数字 1 表示单用户模式，单用户模式类似于 Windows 系统中的安全模式。当系统的运行级别切换至 1 时，系统只允许 root 用户登录，单用户模式一般用于对系统进行维护。
- 2：多用户模式，当系统处于运行级别 2 时，用户不能使用 NFS（网络文件系统）。在运行级别 2 之下系统将会拒绝向网络中的其他计算机提供服务，此模式一般用于维护系统。
- 3：完全多用户模式：完全多用户模式是 Linux 系统在命令行模式中正常工作的运行级别，目前许多服务器都使用这一运行级别。
- 4：未分配使用。此级别主要由开发人员定制其功能，目前主要用于单片机或其他

系统（例如手机操作系统）的开发和应用。

- 5：图形模式。这一运行级别和运行级别 3 基本相同，不同的是该模式下用户将使用图形界面登录并使用 Linux 系统。
- 6：重新启动。在这一运行级别下系统会立即重新启动。

技巧：如果忘记 root 用户密码，可以在系统启动时，将系统的运行级别切换到单用户模式，然后再重新设置 root 用户密码。

（2）使用 init 命令关闭、重启系统

使用 init 命令立即关闭系统：

```
# init 0
```

使用 init 命令立即重启系统：

```
# init 6
```

使用 init 命令时，系统将会立即执行切换运行操作，因此应该先查看是否有其他用户正在使用系统。

2.4　Linux 命令基础及帮助

使用 Linux 系统过程中，难免会遇到各种各样的问题，这些问题可能是不知道应该用什么命令完成某个任务、不清楚命令的选项和格式等。这时可以查询 Linux 系统自带的各种帮助文档。本节将介绍 Linux 系统中命令的格式、查询系统帮助等内容。

2.4.1　Linux 系统中的命令

在使用和管理 Linux 系统过程中，会使用到许多命令，本小节将简单介绍 Linux 中的命令格式，以及如何使用命令等内容。

1．命令的基本格式

几乎所有的 Linux 命令都具有相似的格式，这个基本格式如下：

```
command [option] [parameter]
```

从上面的基本格式中可以看出，Linux 系统中的命令由命令字（command）、选项（option）和参数（parameter）3 部分组成。

注意：Linux 系统中的命令字同 Windows 一样，通常是一些可执行文件（有时也存在一些可执行的脚本）。与 Windows 不同的是，Linux 系统对大小写敏感，所以在输入命令字时一定要注意命令字的大小写。

（1）命令选项可以细化命令的功能，命令的选项可以是一个或多个字母、单词。使用

选项时需要注意以下事项。

- ❑ 选项对大小写敏感，有时命令对同一个字母的大小写都有不同的含义，所以使用字母作为选项时应该特别注意大小写。
- ❑ 命令的多个选项中可能有些是互斥的，不能放在一起使用，因此使用时应该先弄清楚选项的作用。
- ❑ 使用选项时应该在选项前使用减号“–”，引用多个选项时可以分开写，也可以合并在一起，例如命令 ls -l –a 与 ls -al 的效果是等同的。
- ❑ 如果选项是一个单词（通常将其称为长格式选项），通常在选项前使用两个减号“--”，例如 ls --help。

使用多个长格式选项或单字母选项与长格式选项混用时，应该将每个选项分开写，以免系统误解，例如 ls --all　--human-readable –l。

小知识： 许多命令的同一个选项都有两种表示方式，即同时使用短格式选项（一个字母）和长格式选项（一个单词），例如长格式选项 all 也可写为 a，使用这种方式的目的是方便用户记忆这些选项。

（2）参数是命令需要操作的对象，许多命令都要求使用参数。使用参数时需要注意以下事项。

- ❑ 如果命令有多个参数，需要使用空格分隔。
- ❑ 经常作为命令参数的有变量、文件名、路径等，使用时应该特别注意格式，以免系统误解。
- ❑ 如果参数是一个字符串，应该将其放入双引号中。

使用命令时，命令字、选项和参数三者之间应该用空格分隔开，如果命令中包含有可能引起系统误解的符号（例如字符串中的空格、分号等），应该将容易引起误解的部分放入引号中。

2．执行命令和分段命令

直接在命令提示符后面输入命令，然后按 Enter 键即可执行。例如：

```
# ls
```

许多时候，命令字加上选项、参数可能会很长，当超过一行时系统会自动换行并继续输入，这样看起来可能会不太方便，这时可以使用反斜杠“\”对长命令进行分段。使用方法是先输入命令，在要分段的位置输入反斜杠并按 Enter 键。此时系统不会认为当前这条命令已经输入完成，将会在下一行使用提示符“>”（称为辅助提示符）让用户输入未输入完的命令。例如：

```
$ ls \
> -l \
> -a
```

上面这条命令等同于 ls -l -a。

3. 执行多个命令

有时想要在一行同时输入多条命令，让这些命令一条接一条地执行，此时可以使用分号“;”分隔多个命令。

例如使用分号输入并执行多条命令：

```
$ echo "What is your name?";echo "My name is Liwei."
What is your name?
My name is Liwei.
```

2.4.2　帮助之 help 命令和选项

Linux 命令众多且功能和选项各异，使用起来可能并不是很方便，为此 Linux 系统提供了 3 种帮助：help 命令和选项、man 手册页、info 信息页。这些帮助文档详细讲解了如何使用命令，以及命令各选项的含义等。本小节将详细讲解如何使用 help 命令和选项。

1. help命令

help 命令是 Bash Shell 内置的一个很简明的帮助命令。直接使用命令 help 可以看到人机交互程序 Bash Shell 的版本号、提供的命令等内容：

```
$ help
GNU bash, version 3.2.25(1)-release (i686-redhat-linux-gnu)
These shell commands are defined internally.  Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.
…………
typeset [-afFirtx] [-p] name[=valu ulimit [-SHacdfilmnpqstuvx] [limit
 umask [-p] [-S] [mode]            unalias [-a] name [name ...]
 unset [-f] [-v] [name ...]         until COMMANDS; do COMMANDS; done
 variables - Some variable names an wait [n]
 while COMMANDS; do COMMANDS; done  { COMMANDS ; }
```

帮助信息显示了 Bash 的版本、如何获得帮助，以及 Bash 中常用的命令选项等信息。

将一个命令作为 help 命令的参数，可以获得命令的详细帮助。例如要获得命令 cd 的帮助信息：

```
$ help cd
cd: cd [-L|-P] [dir]
    Change the current directory to DIR.  The variable $HOME is the
    default DIR.  The variable CDPATH defines the search path for
the directory containing DIR.
    ……
    The -P option says to use the physical directory structure
    instead of following symbolic links; the -L option forces symbolic links
    to be followed.
```

2．help选项

许多命令都提供了一个帮助选项 help，利用这个选项可以获得命令的用法、选项等帮助信息。例如要获得命令 bind 的帮助信息：

```
$ date --help
Usage: date [OPTION]... [+FORMAT]
  or:  date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
Display the current time in the given FORMAT, or set the system date.

  -d, --date=STRING         display time described by STRING, not `now'
  -f, --file=DATEFILE       like --date once for each line of DATEFILE
……
O to use the locale's alternate numeric symbols if available.

Report bugs to <bug-coreutils@gnu.org>.
```

许多时候用户可能会不记得命令的用法及选项，使用 help 选项可以快速查看命令的用法和选项列表。help 选项是查看命令帮助最简单、快捷的方法，熟悉 Linux 系统的人们经常用到。

2.4.3　帮助之 man 手册

man 手册是一个很详细的帮助手册，其中包括了系统配置文本、用户命令、库函数等帮助内容。本小节将简单介绍 man 手册的使用方法。

提示：许多时候 man 手册也被称为手册页。

手册页提供了命令、配置文本、库函数等多方面的帮助，在手册页中使用数字标识文档的用途如下。

- 1：用户命令，所有的用户都可以查阅并使用此手册中的内容。
- 2：系统调用，查看系统提供的相关调用，通常这部分内容是提供给程序员使用的。
- 3：库函数。
- 4：设备文件。
- 5：文件格式描述，通常是一些配置文件的帮助手册。
- 6：游戏。
- 7：其他杂项。
- 8：只能由 root 用户使用的管理命令和工具等。
- 9：其他。

用户可以通过手册页中的数字标识了解文档的用途。

（1）要查看一个命令的手册页，可以将命令作为 man 命令的参数直接运行。例如要查看命令 passwd 的 man 手册：

```
# man passwd
PASSWD(1)                    User utilities                    PASSWD(1)
```

```
NAME
      passwd - update a user’s authentication tokens(s)
……

       #
       # passwd service entry that does strength checking of
       # a proposed password before updating it.
:
```

这是一个交互式文档，其常用的快捷键如下。

- ❑ Page up/Page down：向前、后翻页。
- ❑ 空格键：向后翻页。
- ❑ 上、下方向键：向上、下翻动一行。
- ❑ Enter 键：查看下一行。
- ❑ /pattern：输入并按 Enter 键，可以在当前手册中查找与 pattern 匹配的字符串。
- ❑ n/N：与/pattern 查找配合使用，查看下一个/上一个找到的字符串。
- ❑ q：退出 man 交互式文档。
- ❑ h：查看帮助。

在上面的示例中，手册开头的 PASSWD(1)表明了所有用户都可以使用这个命令。

有些命令只供管理员使用，例如下面这个例子：

```
# man ifconfig
IFCONFIG(8)              Linux Programmer’s Manual              IFCONFIG(8)

NAME
      ifconfig - configure a network interface

SYNOPSIS
      ifconfig [interface]
      ifconfig interface [aftype] options | address ...

……
Address Families
      If the first argument after the interface name  is  recognized  as  the
:
```

这是一个供管理员使用的帮助手册，因此 ifconfig 命令的标识为数字 8。

（2）man 手册页也可以指定用途标识查看帮助。例如要查看 read 函数在编写程序时的帮助信息：

```
$ man 2 read
READ(2)                  Linux Programmer’s Manual                  READ(2)

NAME
      read - read from a file descriptor
……
      error  if  this  number  is smaller than the number of bytes requested;
```

```
          this may happen for example because fewer bytes are actually  available
          right  now  (maybe  because we were close to end-of-file, or because
we
:
```

（3）man 手册不仅涵盖了命令，还包括一些配置文件。例如要查看 passwd 文件的帮助手册：

```
# man 5 passwd
PASSWD(5)                   Linux Programmer's Manual                    PASSWD(5)

NAME
          passwd - password file
……
          the  encrypted  passwords  are  in  /etc/shadow  which   is   readable   by
the
          superuser only.

          Regardless  of whether shadow passwords are used, many sysadmins use
          an
:
```

（4）如果在使用 Linux 的过程中忘记了某个命令，或者需要完成某个特定的任务不知道应该使用什么命令，可以使用一个关键字并配合 k 选项进行查找。

例如需要查找所有与网络相关的命令及简介：

```
# man -k network
Net::Cmd                 (3pm)   - Network Command class (as used by FTP, SMTP etc)
Net::Time                (3pm)   - time and daytime network client interface
NetworkManager            (8)   - network management daemon
NetworkManager           (rpm) - Network connection manager and user applications
……
```

可以从其结果中查找相关的命令，并阅读其详细的说明，然后再使用相应的命令完成任务。

与 help 命令和选项相比，man 手册提供了更详细的帮助文档，当忘记命令、选项的作用时，可以使用 man 手册查看详细帮助信息。

2.4.4　帮助之 info 信息页

info（通常将其称为信息页）与手册页一样，也是一个交互式的帮助文件，不同的是信息页采用章节式结构，并且使用了与网页一样的超链接。本小节将简单介绍信息页的使用方法。

信息页的基本操作与手册页相同，除基本操作外，信息页还有超链接的相关操作，这些操作的快捷键和功能如下。

- Tab：跳转到下一个超链接。
- n：跳转到下一节。
- p：跳转到上一节。

- u：跳转到上一层章节。
- Enter 键：当光标处于超链接上时，将跳转到超链接指向的位置。
- ?：查看帮助。

在信息页中使用星号“*”表示超链接，将光标停留在“*”号上，按 Enter 键即可跳转到超链接指向的位置。

（1）查看与 pwd 相关的信息页如下：

```
$ info pwd
File: coreutils.info,  Node: pwd invocation,  Next: stty invocation,  Up:
Worki\
ng context

19.1 `pwd': Print working directory
……
   An exit status of zero indicates success, and a nonzero value
indicates failure.

--zz-Info: (coreutils.info.gz)pwd invocation, 20 lines --All-------------
Welcome to Info version 4.8. Type ? for help, m for menu item.
```

这里可以看到命令 pwd 的信息页为 19.1 章节。此时按 n 键即可跳转到下一个章节：

```
File: coreutils.info,  Node: stty invocation,  Next: printenv invocation,
Prev\
: pwd invocation,  Up: Working context

19.2 `stty': Print or change terminal characteristics
=====================================================
……
terminal line operation, as described below.

   The program accepts the following options.  Also see *Note Common
--zz-Info: (coreutils.info.gz)stty invocation, 71 lines --Top-------------
```

如果连续多次按 n 键，可以继续向下跳转，直至 19 章结束；如果按 p 键则可以返回前一个小节。

（2）按 Tab 键可以在超链接之间跳转。例如要查看超链接“stty invocation:”，按 Tab 键将光标停留在“stty invocation:”前面的“*”号上，按 Enter 键即可：

```
File: coreutils.info,  Node: stty invocation,  Next: printenv invocation,
Pre\
v: pwd invocation,  Up: Working context

19.2 `stty': Print or change terminal characteristics
=====================================================
……
   The program accepts the following options.  Also see *Note Common
--zz-Info: (coreutils.info.gz)stty invocation, 71 lines --Top-------------
```

info 信息页提供的帮助比手册页更加详细和系统，而手册页通常对某个命令或某方面解释得很详细，二者各有长处，用户可以按需要选择使用。

注意：由于许多命令选项、文档等都十分复杂且不常用，因此经常使用这些帮助是一个十分好的习惯。此外互联网上有许多 Linux 社区，用户遇到困难时，也可以向这些社区求助。

2.5　系统与用户的交互程序 Shell

在第 1 章中讲到 Linux 发行版由内核、人机交互程序及其他应用程序组成，其中人机交互程序主要负责将用户的输入交给内核，再将内核的执行结果返回给用户。人机交互程序的交互方式可以是命令行模式，也可以是图形界面，例如 Windows 就是图形化的人机交互界面。

在命令行模式中，负责与用户交互的人机交互程序称为 Shell（Shell 这个词翻译成中文是“外壳”的意思，它自身的含义就说明了其与 Linux 内核之间的关系）。Shell 是一个非常特殊的程序，包含在内核之外，主要作用是将用户输入的命令解释成内核能够识别的指令，然后传递给内核，内核控制硬件执行相关的指令并将结果通过硬件输出。从这个过程可以看出 Shell 的主要作用是负责人与内核之间的交互。

默认情况下，Linux 系统中会安装多个 Shell，系统中的每个用户都可以使用这些 Shell。当用户被创建时，系统会为用户指定一个默认的 Shell，如果没有指定使用何种 Shell，大多数 Linux 会指定 Bash Shell 作为用户的默认 Shell。本节将简单介绍 Linux 系统中的人机交互程序 Shell。

2.5.1　Shell 分类

Linux 系统中安装了多个 Shell，这些 Shell 功能各异，在用法上也大相径庭。对于用户而言，可能并不了解这些 Shell，甚至不知道这些 Shell 在功能上有何长处。本小节将简单介绍一些比较流行的 Shell 及其特性。

1. Bourne Shell

Bourne Shell 简称 Bsh。Bsh 是最早的 UNIX Shell，1979 年 Bourne Shell 诞生于贝尔实验室，编写它的作者是 Stephen Bourne，它由此而得名。它不仅可以解释命令，还可以用来编写脚本，尽管在功能上有一些缺憾（例如缺少进程控制等），到目前为止仍有很多人在使用它。

注意：几乎所有的 UNIX 和 Linux 都安装了 Bsh，不仅如此，在某些特殊情况下（例如救援模式下），Bourne Shell 可能是唯一可用的 Shell。

2. C Shell

C Shell 简称 Csh，诞生于 20 个世纪 80 年代，由加州大学的 Berkeley 分校开发。熟悉互联网的读者可能对这个学校并不陌生，大名鼎鼎的域名服务器软件 BIND 就是该校发展

的 BSD UNIX 中的一部分。在 Shell 提示符下进行操作时可能并不会发现 C Shell 与其他 Shell 的区别，然而在使用 Csh 编写脚本时，可以发现 Csh 与 C 语言的语法结构很相似（因此而得名）。

除了 Csh 外，大多数发行版中还安装有另一个 Tcsh，这是一个增强版的 Csh，在 Csh 的基础上扩展了许多新的功能。

3. Korn Shell

Korn Shell 简称 Ksh，编写它的作者是 David Korn，并以 Korn 命名。Ksh 与 Bourne Shell 一样，也来自贝尔实验室。Ksh 对 Bourne Shell 进行扩展的同时，还集成了许多来自 C Shell 的特性，新增了数学运算、行内编辑等功能，正因如此，Ksh 很受欢迎。

4. Bourne Again Shell

Bourne Again Shell 简称 Bash，它是 GNU 的一个项目，其最初的目的是为基于 GNU 的操作系统编写的标准 Shell。Bash 是目前最为流行的 Shell（几乎所有的 Linux 发行版都使用 Bash 作为默认的 Shell），它能够很好地兼容 Bsh，几乎所有用 Bsh 编写的脚本都能在 Bash 中运行。

5. Z Shell

Z Shell 简称 Zsh，出现在 20 世纪 90 年代初期。与其他的 Shell 相比，Z Shell 拥有强大的功能，融合了众多 Shell 的优点。虽然如此，Zsh 的使用者却很少，其原因在于强大的功能背后，是众多的选项和复杂的配置。为了能有一个十分有个性而功能强大的 Zsh，用户不得不花费更多的时间配置它，这与多数使用者简单而实用的初衷相违背。

除了以上这些 Shell 外，还有诸如 Ash 等其他一些 Shell，感兴趣的读者可以阅读相关资料。虽然 Linux 系统中有许多功能各异的 Shell，但对于大多数用户而言，只需要熟练地使用其中一种就可以了。

注意：如果没有特殊说明，本书中的命令、脚本运行的环境都是 Bash。

2.5.2　更改默认 Shell

虽然 Linux 使用 Bash 作为默认 Shell，但有时可能需更改默认的 Shell。本小节将简单介绍如何更改默认 Shell。

1. 查看系统中安装的Shell及版本

使用默认的 Bash 以外的 Shell 时，应该首先查看系统上安装了哪些 Shell。系统将安装的 Shell 程序列表放在/etc/shells 文件中，可以通过查看该文件获取 Shell 列表：

```
# cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
```

```
/bin/tcsh
/bin/csh
/bin/ksh
/bin/zsh
```

上面的命令中，cat 命令的主要作用是查看文本文件的内容，整个命令运行的结果是列出系统中已经安装的所有 Shell。

注意：/sbin/nologin 并不是一个可用的 Shell，当用户使用它作为默认 Shell 时，系统将会阻止用户登录系统。

有些使用者可能更喜欢某些版本的 Shell，这时就需要查看相关 Shell 的版本信息。几乎所有的 Shell 都支持 version 选项，可以使用这个选项查看 Shell 的版本信息：

```
# bash --version
GNU bash, version 3.2.25(1)-release (i686-redhat-linux-gnu)
Copyright (C) 2005 Free Software Foundation, Inc.
# sh --version
GNU bash, version 3.2.25(1)-release (i686-redhat-linux-gnu)
Copyright (C) 2005 Free Software Foundation, Inc.
# csh --version
tcsh    6.14.00    (Astron)    2005-03-25    (i386-intel-linux)    options
wide,nls,dl,al,kan,sm,rh,color,filec
# tcsh --version
tcsh    6.14.00    (Astron)    2005-03-25    (i386-intel-linux)    options
wide,nls,dl,al,kan,sm,rh,color,filec
# ksh --version
  version         sh (AT&T Research) 93s+ 2008-01-31
# zsh --version
zsh 4.2.6 (i686-redhat-linux-gnu)
```

在上面的版本信息中，sh 和 bash、csh 和 tcsh 版本信息相同。

利用下面的命令可以查看 Shell 程序的相关信息：

```
# ls -l `cat /etc/shells`
-rwxr-xr-x 1 root root  735004 Oct 21  2008 /bin/bash
lrwxrwxrwx 1 root root       4 Jul 26 04:40 /bin/csh -> tcsh
-rwxr-xr-x 1 root root 1089956 Oct  1  2008 /bin/ksh
lrwxrwxrwx 1 root root       4 Jul 26 04:36 /bin/sh -> bash
-rwxr-xr-x 1 root root  345764 Sep  2  2008 /bin/tcsh
-rwxr-xr-x 1 root root    5120 Nov 25  2008 /sbin/nologin
```

上面使用的命令 ls -l `cat /etc/shells`，其中的反引号表示将引号内的命令输出以变量的形式交给 ls 命令。

从上面的文件信息可以看出 sh、csh 分别是 bash 和 tcsh 的符号链接文件（与 Windows 中的快捷方式类似），这样做是为了保证向下的兼容性。

2．更改用户默认Shell

了解以上信息之后就可以更换用户 Shell 了，要更改用户 Shell，一般有两个办法：临时修改当前使用的 Shell、修改用户默认 Shell。

（1）临时修改当前使用的 Shell

如果某个系统在使用过程中出现了一些问题，通过管理员授权，另一个用户（可能是更有经验的维护人员）使用管理员的授权账号登录系统。授权用户可能并不习惯使用该账号默认的 Shell，这时可以临时使用系统上已安装的其他 Shell 维护系统。

要临时更改当前使用的 Shell，可以直接输入需要使用的 Shell 命令。例如要临时使用 Ksh Shell：

```
[user1@localhost ~]$ ksh
$ echo $SHELL
/bin/bash
$ exit
[user1@localhost ~]$
```

在上面这个临时更改 Shell 的例子中，命令 echo $SHELL 输出的结果显示当前用户使用的仍然是 Bash。原因是当前正在使用的 Ksh 是运行在 Bash 下的一个程序，即 Bash 将 Ksh 看做一个程序来运行（此时一般称 Ksh 是 Bash 的子 Shell）。要避免 Ksh 成为 Bash 的子 Shell，只有通过修改用户默认 Shell 的方法实现。

提示： 如果要退出临时的 Ksh，可以使用 exit 命令，大多数 Shell 都支持该命令。不支持这个命令的 Shell 可以使用快捷键 Ctrl+C，强制结束被认为是应用程序的子 Shell。

（2）修改用户默认 Shell

如果用户想要自己修改默认 Shell，可以使用 chsh 命令。例如要修改用户的默认 Shell 为 Ksh：

```
#使用 grep 命令查看/etc/passwd 文件中包含 user1 的行
[user1@localhost ~]$ grep user1 /etc/passwd
#从下面的内容可以看出 user1 使用的默认 Shell 为 Bash
user1:x:500:500::/home/user1:/bin/bash
[user1@localhost ~]$ chsh
Changing shell for user1.
#按提示输入当前用户的密码
Password:
#输入默认 Shell
New shell [/bin/bash]: /bin/ksh
Shell changed.
[user1@localhost ~]$ grep user1 /etc/passwd
user1:x:500:500::/home/user1:/bin/ksh
```

chsh 命令要求输入当前的密码和更改后的 Shell，正确输入后就可以完成修改了。从命令的执行结果可以看出，chsh 通过修改文件/etc/passwd 的 Shell 字段的方式修改默认 Shell。

上面的命令中使用了一个系统文件/etc/passwd，该文件用于保存系统中所有的用户及其设置，该文件的内容将在 6.1 节中介绍。

然而 chsh 命令并不能被所有系统支持，这时就需要求助于系统管理员。系统管理员修改时可以使用 usermod 命令：

```
# grep user1 /etc/passwd
user1:x:501:501::/home/user1:/bin/bash
```

```
#使用 usermod 命令将用户 user1 的默认 Shell 改为 Ksh
# usermod -s /bin/ksh user1
# grep user1 /etc/passwd
user1:x:501:501::/home/user1:/bin/ksh
```

修改完成后，用户只要重新登录系统就可以使用修改后的默认 Shell 了。

如果要在用户创建时指定其默认 Shell，可以使用：

```
# useradd -s /bin/ksh user2
# grep user2 /etc/passwd
user2:x:505:505::/home/user2:/bin/ksh
```

上面的命令将添加一个名为 user2 的用户，该用户的默认 Shell 为 Ksh。

2.6　Bash 中的命令基本操作

最早的 Bash 是从 Bsh 的基础之上发展而来的，通过多次修正和补充，吸收了众多 Shell 的特性。Bash 的众多特性使其成为大多数 Linux 发行版的默认 Shell，这也是其成为当前最为流行的 Shell 的原因之一。除此之外，Bash 还拥有众多的优点，下面列举了 Bash 的一些优点。

- 命令行编辑功能：使用户可以在输入命令时使用方向键前后移动光标，随意执行编辑操作。
- 历史命令功能：使用户可以查看之前输入的命令。历史命令功能便于用户查找导致系统出现问题的命令，也方便用户重复输入同一条命令。
- 命令别名功能：用户可以定义自己的命令别名。使用此功能不仅可以将复杂的命令减化输入，还可以按照用户的个人爱好定制属于用户自己的“命令”。
- 命令和文件名补全：命令和文件名补全功能不仅能方便用户输入命令和文件名，还大大降低了记忆命令的难度。
- 变量和流程控制：Bash 提供的变量和流程控制功能，大大提高了脚本编程的灵活性。

除此之外，Bash 还有许多优点（例如强大的脚本编程功能等），此处不一一介绍，感兴趣的读者可以阅读相关文档了解。

Bash 的众多优点使其成为当前最流行的 Shell，这也是本书将 Bash 作为主要内容的重要原因。了解了 Shell 和 Bash 的优点之后，本节将介绍 Bash 中的命令基本操作。

2.6.1　命令行编辑功能

命令行编辑功能是指用户可以使用方向键前后移动光标，并编辑已经输入的命令，这个功能非常实用、方便。本小节将简单介绍命令行编辑功能的使用方法。

提示：可能大多数读者都没有使用过不带命令行编辑功能的 Shell，有兴趣的读者可以使用系统自带的 Bsh。Bsh 是一个不带命令行编辑功能的 Shell，输入命令时如果发现某处输入错误，不能修改，只能重新输入该命令，非常不方便。

（1）更改行编辑的首选编辑模式

Bash 同时支持 Vi 和 Emacs 编辑器（Vi 和 Emacs 是 Linux 系统中最常用的两种编辑器）中的一些行编辑快捷键，如果未进行设置，Bash 将使用 Emacs 编辑器的风格。如果需要修改首选编辑模式，可以使用命令 set -o 指定。

例如修改首选编辑模式为 Vi：

```
# set -o vi
```

运行上面的命令之后，Bash 将使用 Vi 的风格接受输入。这时就可以使用 Vi 编辑器的方式进行操作了，例如使用 H 和 K 前后移动光标等。

（2）快速移动光标

在命令行中输入命令时，可以使用左、右方向键在字符间快速前后移动光标并修改已输入的命令。移动光标还可以使用以下快捷键：

- Ctrl+B：向前移动一个字符。
- Ctrl+F：向后移动一个字符。
- Ctrl+A：快速移动到行首。
- Ctrl+E：快速移动到行尾。

（3）行内删除

将光标移动到要修改的位置后，可以使用退格键（Backspace）删除光标所在位置的前一个字符，使用删除键（Delete）将删除当前光标处的字符。删除操作的其他快捷键如下所示：

- Ctrl+D：删除当前光标处的字符（相当于 Delete 键）。
- Ctrl+H：删除当前光标的前一个字符（相当于 Back Space 键）。
- Ctrl+U：删除当前光标到行首的所有字符。
- Ctrl+K：删除当前光标到行尾的所有字符。

用户在输入命令时，可以使用这些快捷键快速定位，并删除、修改命令行中已输入内容。

虽然此处介绍了许多行编辑功能，但建议初学者养成自己的习惯即可，不必在学习初期就使用所有的快捷键。

2.6.2　绑定快捷键和命令

命令行模式中默认定义了许多快捷键，这些快捷键由一个名为 Readline 的库实现。用户可以通过命令 bind 添加新快捷键，或修改系统中已经存在的快捷键。本小节将简单介绍如何自定义快捷键。

提示：本小节中的部分知识涉及系统登录环境中的内容，读者可以结合第 16 章进行学习。

1．查看系统中已绑定的快捷键

可以通过 bind 命令的选项 P 查看已经设定好的快捷键：

```
# bind -P
abort can be found on "\C-g", "\C-x\C-g", "\e\C-g".
accept-line can be found on "\C-j", "\C-m".
alias-expand-line is not bound to any keys
arrow-key-prefix is not bound to any keys
backward-byte is not bound to any keys
backward-char can be found on "\C-b", "\eOD", "\e[D".
...
yank can be found on "\C-y".
yank-last-arg can be found on "\e.", "\e ".
yank-nth-arg can be found on "\e\C-y".
yank-pop can be found on "\ey".
```

在上面的命令输出中，“\C”表示 Ctrl 键，例如“\C-b”表示 Ctrl+B 组合键。“\M”表示 Meta 键，另一些以“\e”开头的是一些功能键或编辑键，例如“\e[2~”表示 Insert 键等。

提示：在 Sun 和其他几家公司生产的某些型号的键盘上存在 Meta 键，键帽上有菱形标志，一般的键盘上可以使用 Alt 或 Esc 键替代。

2．绑定快捷键命令bind的基本格式

bind 命令用于定制系统快捷键，其基本格式如下：

```
bind [option] [[function name][keyseq:shell-command][file name][keyseq:
readline-function or readline-command]]
```

bind 命令的常见选项如下。

- keyseq:readline-function：这是一个基本用法，参数 keyseq 表示指定的快捷键，readline-function 参数表示要绑定的函数名称。
- m keymap：使用参数 keymap 指定的键盘映射。
- l：列出所有可用的 Readline 函数名称。
- P：列出所有快捷键列表。
- p：列出当前 Readline 库中的函数名称及绑定。
- f filename：从文件 filename 中读取绑定。
- r keyseq：取消由 keyseq 快捷键指定的所有绑定。
- x keyseq:shell-command：为 keyseq 指定的快捷键绑定由 shell-command 指定的 Shell 命令。

3．修改或自定义快捷操作

在修改或定义快捷键之前，需要先查看 Readline 库中可以使用的函数名称，查看函数名称可以配合使用 l 选项：

```
# bind -l
abort
accept-line
alias-expand-line
```

```
arrow-key-prefix
backward-byte
backward-char
backward-delete-char
...
vi-yank-to
yank
yank-last-arg
yank-nth-arg
yank-pop
```

这些函数都是系统默认提供的，用户可以为这些函数绑定快捷键。

例如将删除光标前一个字符的函数（退格键 Back Space）绑定到快捷键 Ctrl+X 上：

```
# bind "\C-x":backward-delete-char
```

在上面的命令中，“\C-x”表示要绑定的快捷键组合，backward-delete-char 则是 Readline 库中的函数。读者可以直接套用这个格式，为 Readline 库中的函数绑定快捷键。

4．绑定应用程序和命令

bind 命令不仅可以绑定一个已有函数，还可以配合使用选项 x 绑定一个程序。绑定程序的目的通常是为了能在需要时快速执行程序。

例如为了方便输入文本，使用快捷键 Ctrl+Alt+V 快速打开 Vi 编辑器：

```
# bind -x '"\C-\M-v":vi'
```

选项 x 不仅可以绑定一个程序，还可以绑定 Shell 命令，因此用户可以使用这种方式加快输入命令。

例如连续按两次快捷键 Ctrl+X 就在当前目录中使用 Shell 命令 ls -l：

```
# bind -x '"\C-x\C-x":ls -l'
#
#按两次 Ctrl+x
total 156
drwxr-xr-x  2 root root 4096 Jul 26 04:41 account
drwxr-xr-x 10 root root 4096 Jul 26 05:00 cache
drwxr-xr-x  2 root root 4096 Dec  7  2006 cvs
...
drwxrwxrwt  2 root root 4096 Jul 31 10:25 tmp
drwxr-xr-x  3 root root 4096 Jul 26 04:40 yp
```

在上面两条使用选项 x 的命令中，都使用了将快捷键及要执行的命令或程序放入单引号中的方法，其目的是避免 bind 误解。

5．保存修改的快捷键

修改或添加快捷键后，如果系统重新启动或用户重新登录，这些设置将会丢失，如果需要继续生效，就要将这些设置保存在配置文件中。

（1）在系统中有两个文件用于保存设置的快捷键，其一是/etc/inputrc，该文件中保存的快捷键对登录系统的每个用户都会生效；另一个文件是~/.inputrc，这个文本中保存的快

捷键只会对家目录对应的用户生效。

要保存已设置的快捷键，可以使用选项 p 输出所有绑定，然后用重定向的方法保存到用户的家目录中：

```
# bind -p > ~/.inputrc
```

完成上述操作后，用户只需要重启系统或重新登录，之前定义的快捷键就会生效。

注意：由于文件/etc/inputrc 中保存的快捷键对每个用户都生效，因此通常不推荐将快捷键保存在此文件中。

（2）然而有的系统不支持上述办法，这时可以尝试执行以下命令：

```
# echo 'export $INPUTRC=~./inputrc' >>~/.bash_profile
```

上面这条命令的作用是将 export $INPUTRC=~./inputrc 追加到文件~/.bash_profile 的结尾（环境变量 INPUTRC 的作用是保存快捷键文件的路径）。

执行以上命令后，系统读取快捷键设置时，将不再读取文件/etc/inputrc 中的内容，而是读取用户家目录中的文件~/.inputrc。

（3）然而不幸的是有的系统对以上两种方法都不支持，这时可以在文件~/.bash_profile 最后加入以下内容：

```
#使用 cat 命令查看文件.bash_profile 的内容
# cat .bash_profile
# .bash profile

# Get the aliases and functions
......
export PATH

#以下为追加的内容
#使用 if 语句判断是否存在文件~/.inputrc
if [ -f ~/.inputrc ]; then
#如果文件存在，则使用 f 选项读取文件中的快捷键设置
       bind -f ~/.inputrc
fi
```

添加了上述内容之后，用户登录时将会主动读取已保存的快捷键设置。

（4）上面的 3 种方法只能保存为 Readline 库中的函数设置的快捷键，自定义的程序和 Shell 命令将不会被保存，使用以下方法可以保存这些自定义的程序和 Shell 命令。

首先在用户主目录下新建一个名为~/.custom_bind 的文件，将要绑定的信息写入其中。这个例子将使用上面的两条命令：

```
#使用 cat 命令查看文件.custom bind 的内容
# cat .custom_bind
bind -x '"\C-\M-x":ls -l'
bind -x '"\C-\M-v":vi'
```

然后编辑用户主目录下的文件.bash_profile，向文件结尾追加以下内容：

```
# cat .bash profile
# .bash profile

# Get the aliases and functions
......
export PATH
#以下为追加的内容
#Additional content here
if [ -f ~/.custom bind ]; then
       . ~/.custom bind &>/dev/null
fi
```

像上面的内容一样，if 语句先判断是否存在文件~/.custom_bind，如果存在，就执行后面的命令。“. ~/.custom_bind &>/dev/null”（点号后面有一个空格）这条命令前面的“.”告诉 Bash 应该像命令那样执行文件~/.custom_bind 的内容。最后的&>/dev/null 表示如果在执行过程中有需要输出的信息（例如命令的正常输出、输出的错误提示等），就将要输出的信息写入到文件/dev/null（/dev/null 是一个字符设备文件，所有输出到这个文件的内容将会被丢弃，所以人们将它形象地称为“系统垃圾池”或“系统池”）中。

提示： 如果要改变所有用户的快捷键设置，仍然不建议将绑定写入文件/etc/inputrc 中。此时可以将相关的文件放入目录/etc/skel/中，并修改该目录中的相关文件（该目录中的内容同家目录中的内容完全一致）。当新用户被创建时，系统会自动将目录/etc/skel/中的内容拷贝至新用户的家目录中，这样新添加的用户就获得了自定义的快捷键设置。

2.6.3　命令行补全功能

当用户输入一个命令或文件名前缀后，按下制表符 Tab 键，Bash 将会自动查找以当前字符为前缀的命令或文件名称。如果找到且结果唯一，Bash 将自动补全命令并在补全的命令后空一格。如果找到结果且结果有多个，则什么也不显示，如果按两次 Tab 键，Bash 会在下一行显示所有匹配前缀的命令或文件名，并重新返回提示符和输入的前缀。这就是 Linux 系统中的命令行补全功能，其操作过程与思科 IOS（思科交换机、路由器专用操作系统）中的操作十分相似。下面从两个方面介绍这种补全功能。

1. 自动补全命令

下面将以命令 service 作演示。输入字符 s 然后按 Tab 键，此时 Bash 无任何返回信息。如果你的 Linux 系统能使用声音设备（可能是声卡，也可能是主板上的蜂鸣器），将会发出提示音，提示音告诉用户匹配这个前缀的命令不止一条。此时按两次 Tab 键，Bash 将发出提示音并返回如下信息：

```
# s
#输入 s 并按两次 Tab 键将显示以下内容
Display all 178 possibilities? (y or n)
```

Bash 返回的提示显示，有 178 个可能的匹配命令，并询问是否要全部显示。如果按 y

键，Bash 会分页显示所有可能匹配的命令。继续输入 se 并按两次 Tab 键，此时 Bash 会发出提示音并显示所有匹配命令：

```
# se
#输入 se 并连按两次 Tab 键
sechecker          semodule_link        sestatus     setmetamode
secon              semodule_package     set          setpcaps
……
semodule_deps      sesearch             setkeycodes setup
semodule_expand sessreg              setleds      setxkbmap
# se
```

这时列出的命令就少了很多，用户可以仔细查看提示并从中找出需要使用的命令。输入 ser 按 Tab 键，这时系统返回完整的命令 service，同时也发出了提示音：

```
#输入 ser 后按 Tab 键将显示以下内容
# service
```

此时虽然返回了正确的命令，但 Bash 并没有在命令后面加一个空格，并且提示音告诉用户，能够匹配以当前补全的内容为前缀的命令不止一条。连按两次 Tab 键，Bash 返回有两条命令匹配当前内容：

```
# service
#输入以上内容后连接两次 Tab 键
service      serviceconf
```

从上面这个演示中可以看出，当用户输入的命令前缀足够长时，系统总是能更精确地补全用户需要使用的命令。

2. 自动补全文件名

下面将以查看 samba 服务的配置文件 smb.conf 为例，演示如何补全文件名。输入查看文件命令 cat /e，此时按 Tab 键系统将会自动补全为：

```
# cat /etc/
```

由于匹配/e 前缀的是一个目录，所以在补全的目录名后面加上了/。

在后面继续输入 sa 后按 Tab 键，系统将会发出提示音，并且没有返回任何信息，这表示/etc 目录中以 sa 开头的文件或目录不止一个。

继续按两次 Tab 键：

```
# cat /etc/sa
#输入以上内容后连接两次 Tab 键
samba/ sasl2/
# cat /etc/sa
```

这时 Bash 返回了两个匹配的文件名称，此时在后面继续输入 m，按 Tab 键系统自动补全目录名 cat /etc/samba/。

如果忘记配置文件名称，可以按两次 Tab 键，Bash 将会返回目录中所有的文件名称：

```
# cat /etc/samba/
lmhosts          smb.conf         smb.conf backup
# cat /etc/samba/
```

直接在/etc/samba/后面输入字符 s，按 Tab 键将会补全配置文件名 smb.conf。

从上面的演示中可以看出，文件名补全功能非常方便用户输入文件和路径，使用此功能将大大提高用户输入文件名和路径的速度。

2.6.4　命令历史功能

Bash 中除附带了命令行补全功能外，还附带了一个非常好用的命令历史功能。用户可以使用它快速地重复输入和执行同一个命令，也可以使用它查找用户执行过的错误命令。本小节将简单介绍如何使用 Bash 提供的命令历史功能。

1. 使用命令历史功能

默认情况下 Bash 会记录用户输入的每条命令，可以使用方向键中的上、下两个键快速调出曾经执行过的命令。在提示符下如果按向上方向键，上一条执行过的命令将会自动出现在提示符后面。如果用户多次按向上方向键，Bash 还将显示更多过去执行过的命令。例如：

```
# pwd
#输入并执行命令 pwd
/home/user1
# cd /etc
#输入并执行命令“cd /etc”
# cd /etc
#按一次上方向键显示刚刚执行过的命令，按 Enter 执行
# pwd
#按两次向上方向键显示出以前执行过的命令，按 Enter 执行
/etc
```

提示：在 Bash 中也可以使用快捷键 Ctrl+P 和 Ctrl+N 代替上、下方向键。

有时可能需要查看更多历史命令，这时可以使用命令 history 查看系统记录的历史命令：

```
# history
  ......
  229  pwd
  230  cd /etc
  231  pwd
  232  cd
  233  history
#
```

在上面这个例子中，命令 history 一共输出了 233 个执行过的命令（默认情况下 history 命令最多可以输出 1000 个）。查询到历史命令之后，可以使用符号“!”引用这些历史命令。例如：

```
# !230;!231
cd /etc;pwd
/etc
```

这条命令使用分号“;”分别引用第 230 条和第 231 条命令。

2．在命令历史中搜索

当历史命令众多时，查找起来可能会相当麻烦，这时可以使用历史命令搜索功能。历史命令搜索功能的快捷键是 Ctrl+R，直接使用该快捷键，命令提示符将变为：

```
(reverse-i-search)`':
```

在提示符“:”（冒号）后面输入要查找命令的内容，Bash 将会动态地将与输入相匹配的内容放在“:”提示符后面。如果找到相关的命令，就可以停止输入，并按 Enter 键执行找到的历史命令：

```
(reverse-i-search)`cat': ls -l `cat /etc/shells`
```

此时直接按下 Enter 键后：

```
# ls -l `cat /etc/shells`
-rwxr-xr-x 1 root root  735004 2009-01-22 /bin/bash
lrwxrwxrwx 1 root root       4 08-15 10:57 /bin/csh -> tcsh
……
```

提示符将变回“#”，并直接执行找到的命令并执行。

3．系统如何记录命令历史

每个用户的家目录中有一个名为.bash_history 的文件，该用户的历史命令都被记录到这个文件中。可以通过直接查看该文件内容的方式查看使用过的历史命令：

```
# tail -5 .bash_history
clear
vi .bash_profile
clear
vi .bash_profile
exit
```

细心的读者可能会发现，查找整个文件都没有找到当前输入的命令。事实上用户输入命令时，Bash 只会将命令记录到系统缓存中，当 Bash 退出时（这个过程发生在用户登出系统或系统关闭时），才会将缓存中的历史命令写入.bash_history 文件中。

4．控制命令历史功能

有时可能需要控制用户的历史命令功能，这时就需要注意到系统中的 3 个环境变量：

```
#使用 set 命令输出所有的环境变量并从中筛选输出含有 HIST 的行
# set | grep HIST
HISTFILE=/home/user1/.bash history
HISTFILESIZE=1000
HISTSIZE=1000
```

上面的变量中定义了保存历史命令的文件和历史命令的条数等内容，用户也可以通过修改这 3 个变量的值，改变 Bash 记录历史命令的位置和条数。

5．清空命令历史

命令历史使用起来非常方便，但也可能会带来一些问题（通常是安全性问题），如果要清空历史命令，可以使用 history 命令的 c 选项：

```
# history -c
# history
    1  history
```

注意：如果系统供多个用户使用，root 用户保存的历史命令可能会引发一些安全性问题。这时可以使用修改环境变量的方法，让 Bash 不记录历史命令。

2.6.5　命令别名功能

在管理和维护 Linux 系统的过程中，将会使用到大量命令，有一些很长的命令或用法经常被用到，重复而频繁地输入某个很长命令或用法是不可取的。这时可以使用命令别名功能将这个过程简单化。

1．系统定义的别名

通常情况下，系统中已经定义了一些命令别名，要查看已经定义的命令别名，可以使用 alias 命令：

```
#alias 命令将输出所有已经定义的命令别名
# alias
alias cp='cp -i'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot
--show-tilde'
```

从上面的结果中可以看出，当我们使用命令 cp（复制文件命令）时，系统会用 cp -i 代替命令中的 cp。除此之外，还定义了 ls 命令及其使用的颜色、移动文件命令 mv、删除命令 rm 等。

用于设置系统别名的相关文件保存在/etc/profile.d/目录中（系统别名目录），使用以下方式可以查看：

```
#进入目录/etc/profile.d/
# cd /etc/profile.d/
#查看目录中的文件
# ls
colorls.csh  glib2.sh                krb5-workstation.csh  lang.sh   vim.csh
colorls.sh  gnome-ssh-askpass.csh  krb5-workstation.sh   less.csh  vim.sh
glib2.csh    gnome-ssh-askpass.sh   lang.csh              less.sh   which-2.sh
#查看文件 less.csh 的内容
# cat less.csh
```

```
#以下为 less.csh 的内容，其中定义了 ls 命令使用的颜色等别名
# less initialization script (csh)
if ( -x /usr/bin/lesspipe.sh ) then
  setenv LESSOPEN "|/usr/bin/lesspipe.sh %s"
endif
# cat colorls.sh
# color-ls initialization

alias ll='ls -l' 2>/dev/null
alias l.='ls -d .*' 2>/dev/null
......
```

2. 用户自定义别名

许多时候管理员都会按自己的使用习惯定义命令别名。例如让查看当前文件内容的命令兼容 DOS 中的查看文本命令 type：

```
#为 cat 命令定义一个别名 type
# alias type='cat'
#使用 type 命令查看文件 alias.txt 的内容
# type alias.txt
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot
--show-tilde'
```

上面的命令中，先为 cat 命令定义了一个名为 type 的别名。当用户使用命令 type 时，系统会自动使用 cat 命令将其替代。

3. 取消定义的别名

要取消已经定义好的命令别名，可以对别名使用 unalias 命令：

```
# unalias type
# type alias.txt
-bash:type: command not found
```

4. 保存别名设置

当系统重新启动或用户重新登录时，使用 alias 命令定义的别名将会丢失。可以在系统别名目录中添加别名配置文件，但这种方式定义的别名对所有的用户都生效，通常不建议使用这种方法。

如果要定义全局别名，通常建议将命令添加到全局配置文件/etc/profile 中。例如定义全局别名：

```
# echo "alias type='cat'">>/etc/profile
```

这条命令将 alias pg='cat'添加到文件/etc/profile 中。

🔔注意：在对/etc/profile 这样的系统配置文件进行操作时，一定要谨慎，否则有可能会损坏系统。因此上面的命令中使用的是“>>”而不是“>”，“>>”表示将内容追加到文件结尾。

如果某个用户想要定义自己的命令别名，可以将命令添加到用户家目录中的文件.bash_profile 中。例如要定义用户自己的别名：

```
# echo "alias vi='vim'">>~/.bash_profile
```

2.7　管道和输入/输出

在管理和维护 Linux 系统的过程中，经常需要对一个命令的输出使用一连串的命令，Bash 提供的管道、输入/输出重定向功能可以轻松地完成这个任务。由于该功能在 Linux 系统中使用非常频繁，因此所有人都应该学会使用这个功能。本节将详细讲解如何使用管道和输入/输出重定向功能。

2.7.1　管道

管道的功能就像这个词本身的含义一样，不同的是现实中的管道用来引导水流，而此处的管道则被用来传递文本信息。本小节将介绍如何使用管道。

（1）管道使用竖线“|”作为操作符，这个符号通常位于键盘上的 Enter 或退格键 Back Space 附近。管道的格式为：

```
# command1 | command2 | comand3 ...
```

系统执行上面这种命令时，会先执行命令 command1，并通过管道将 command1 的输出结果交给命令 command2 处理。然后再将命令 command2 的结果传给下一个命令 command3，这样一直将结果传递下去，直到最后一个命令。在这个过程中，管道后面的命令总是使用管道前面的命令输出作为命令的输入。

（2）许多时候需要在某个命令的输出中筛选结果，这是管道最常见的用法之一。例如查看系统中是否已安装了 gcc 编译器软件包：

```
# rpm -aq | grep gcc
gcc-gfortran-4.1.2-44.el5
gcc-c++-4.1.2-44.el5
libgcc-4.1.2-44.el5
gcc-4.1.2-44.el5
```

上面这条命令中，先使用命令 rpm -qa 输出系统中已安装的所有软件包，然后利用管道将结果传递给下一条命令 grep gcc。后者将从已安装的软件包列表中筛选出名称中含有 gcc 的行并输出。

（3）管道的另一个常见用法是辅助处理命令输出，例如要输出本地网络接口卡收发数据详情：

```
# netstat -i | sed '1d' | awk '{OFS="\t"}{print $1,$4,$8}'
Iface    RX-OK    TX-OK
eth1     845      64
eth2     1375     1435
lo       8        8
```

上面这条命令中，先使用 netstat -i 输出本地网卡的使用情况，由于这个命令输出了很多我们不需要的内容，因此要对这个命令输出进行编辑。先使用管道将网卡使用情况传递给下一条命令，sed '1d'将会删除网卡使用情况文本的第 1 行。然后再一次使用管道，将 sed 命令的输出结果传递给下一命令。命令 awk 收到传递来的输出后，首先按空格键或制表符 Tab 分隔每一行，然后再输出第 1 个、第 4 个和第 8 个字段，并在每个字段间用制表符 Tab 隔开。

管道是 Bash 的重要功能之一，使用管道可以将一个命令的输入传递给另一个命令，十分方便用户处理命令输出。本书的后续章节中还有更多使用管道的例子，此处不再一一举例。

注意：虽然一条命令中可以使用多个管道符，但是不能在一条命令中执行多次对文件的读写操作。

2.7.2　命令的输入/输出和错误

在执行命令时，可能希望命令从某个文件中读取输入，或将命令的输出写入到文件中，这时就要使用到重定向功能。在了解重定向功能之前，需要了解 Linux 如何处理命令的输入/输出和错误。

在 Linux 系统中，执行每个命令系统都会访问 3 个文件，这 3 个文件分别用于接收用户的输入、输出命令的结果和输出命令错误信息。访问文件又是通过文件描述符的方式实现的，命令访问的 3 个文件及其文件描述符如下所示。

- 标准输入（默认为键盘）：标准输入使用的是文件描述符 0。
- 标准输出（默认为显示器）：标准输出使用的文件描述符为 1。
- 标准错误（默认为显示器）：用文件描述 2 表示标准错误。

提示：文件描述符的相关概念，读者可以自行参考 C、C++等编程语言的相关内容。

形象地说这些描述符就像一个自来水厂的管道，一端固定指向自来水厂的入水管（输入）、出水管（输出）和废水管（错误），另一端则默认指向水源地（键盘）、用户管网（显示器）和回收水池（显示器）。当改变水管的指向时，就从指向的目的地抽水（读取数据）或将水输送给目的地（保存数据）。因此可以通过改变这 3 个文件描述符指向的方式，从其他文件或设备中读取或输出数据，此时即称为重定向标准输入/输出和错误。

2.7.3　重定向命令的输入/输出和错误

在 Linux 中，重定向功能用途十分广泛，例如保存命令输出、屏蔽命令输出、分离命令输出等，因此初学者应该学会使用重定向功能。本小节将介绍如何使用重定向功能。

1．输出重定向

在管理和维护系统的过程中，经常需要将一个或多个组合命令的输出结果保存到文件中，以便于分析系统的状况等信息，这时就要使用到重定向输出。重定向输出使用的操作符是“>”和“>>”，这两个操作符表示将结果写入、追加到指定的文件中。

例如要将网络信息保存在当前目录中的 netstat.txt 文件中：

```
# netstat -tunlp | sed '1,2d' | awk '{OFS="\t"}{print $6,$7}' >netstat.txt
# cat netstat.txt
LISTEN  3683/portmap
LISTEN  3021/cupsd
LISTEN  4004/sendmail:
LISTEN  3713/rpc.statd
LISTEN  3978/sshd
4159/avahi-daemon:
……
```

上面这条命令可以将系统服务使用的商品信息保存到文件中，以便于管理员分析。

另一个重定向的例子是使用“>>”操作符将内容追加到指定的文件结尾，而不是覆盖原有文件内容，这个过程就像日志文件的添加方式。

例如将当前的时间追加到文件 netstat.txt 的最后一行：

```
# date>>netstat.txt
```

这条命令执行的结果是将 date 命令的输出结果追加到文件 netstat.txt 的最后一行。

提示：如果要向系统配置文件（特别是/etc 目录中的文件）添加内容，通常应该使用追加的方式，以免损坏系统。

2．输入重定向

要对一个命令输入很长的文本总是很困难的，这时可以考虑将文本输入一个文件，然后使用输入重定向的方式将文本作为命令的输入。输入重定向使用的操作符是“<”和“<<”，大多数时候输入重定向使用的是一个文件。

例如要将统计到的网络接口信息以邮件的形式发送给用户 user1：

```
# mail -s 'Network status' user1 <netstat.txt
```

另一个输入重定向符是“<<”，这个操作符被形象地称为 here document（这儿文档），下面以一个示例进行讲解：

```
# cat <<END
> This is the first line.
> This is the second line.
> This is the last line.
> END
This is the first line.
This is the second line.
This is the last line.
```

在上面这条命令中，cat 命令用于输出文本，“<<END”操作符后面的字符 END 指示了结束输入的标记。按 Enter 键之后用户就可以开始输入文本，直到输入的字符中出现 END，输入过程才会结束。

重定向输入时，如果使用的操作符是“<<-”（而不是“<<”），Bash 会忽略行首的第 1 个空格或制表符。

3．错误重定向

执行一个命令时可能会有很多输出，由于命令的输出信息和错误信息混合在一起，可能无法看到错误信息，这时可以将标准错误重定向到文件中，以便于用户查看。

例如将错误重定向到文件：

```
# rm abcd
#当使用 rm 命令删除一个不存在的文件时，命令将提示错误信息
rm: cannot lstat `abcd': No such file or directory
# rm abcd 2>error
# cat error
rm: cannot lstat `abcd': No such file or directory
```

从上面的演示中可以看到，重定向功能成功地将错误信息写入指定的文件中。当某个命令产生许多输出时，可以使用错误重定向的方法，将错误信息从众多命令输出中分离出来。

默认情况下，重定向符“<”和“>”表示重定向输入和输出，因此在重定向错误时必须使用“2>”明确表示将标准错误进行重定向。如果用户愿意，也可以在重定向输入/输出时，使用数字 1、2 指明要重定向的对象。

技巧： 当我们不想看到错误信息时（例如在系统启动脚本中运行某个命令可能会产生错误），可以将错误信息重定向到系统垃圾池/dev/null。

有时某条命令会产生很多条错误，为了不让后一条错误信息覆盖掉前面记录的文本，此时应该使用“>>”将错误追加到文件最后。例如：

```
# ls -l 2>>~/error
```

4．使用多个重定向

在制作系统监控脚本时，可能需要检测某个关键命令的执行状态，而这个命令的输出对我们毫无用处，此时就需要同时重定向命令的输出和错误。

例如需要检测两台主机间的连通性，并将连通性测试的结果放入文件 netstat 中：

```
# ping 192.168.1.1 -c3 &>/dev/null;echo $? `date` >netstat
# cat netstat
0 Wed Aug 18 13:08:11 PDT 2010
```

上面这几个命令中，先使用 ping 命令发送 3 个数据包测试与另一主机间的连通性（这期间可能会返回错误，也可能会返回连通性测试的结果），然后使用“&>/dev/null”将命令的输出和错误一起定向到系统垃圾池。

上面的例子中，变量“$?”保存的是上一个命令的退出状态，数字 0 表示前一个命令执行成功，非零数字则表示前一个命令执行出现了错误。

技巧：制作系统监控脚本、自动任务脚本时，经常需要将可预见的命令输出和错误重定向到系统垃圾池中。这样做的目的是避免产生的输出、错误干扰用户正在进行的工作。

也可以同时重定向命令的输入和输出，例如：

```
$ sed '6,13d' <netstat.txt >netstat_1
$ cat netstat_1
LISTEN  3683/portmap
LISTEN  3021/cupsd
LISTEN  4004/sendmail:
LISTEN  3713/rpc.statd
LISTEN  3978/sshd
```

将命令的输出和错误同时重定向是管理和维护 Linux 系统的过程中最常见的用法之一，初学者应该特别注意。

2.8　小　　结

- 2.1 节讲解了本书的知识结构、约定和学习 Linux 系统的建议等内容。
- 2.2 节介绍了如何登录 Linux 系统等内容。对于初学者而言，直接在虚拟机界面中学习可能会存在诸多不便，这时建议通过远程登录的方式登录到虚拟机，然后进行学习。
- 2.3 节简单介绍了如何关闭和重启 Linux 系统。其中介绍了许多用于关闭 Linux 系统的命令，在实际应用时，应该根据实际情况选择合适的命令关闭和重启系统。
- 2.4 节简单讲解了 Linux 系统中获取帮助的 3 种方式。通常建议初学者多使用系统提供的帮助，以便于更快地熟悉系统。
- 2.5 节介绍了 Linux 系统中使用的人机交互程序 Shell。虽然 Linux 系统中安装了众多 Shell，但对于大多数用户而言，会熟练使用其中一种 Shell 即可。
- 2.6 节讲解了 Linux 系统中默认的 Bash 的基本操作。命令行补全功能、历史命令功能是这一节的重点，因此初学者需要重点学习这些内容。
- 2.7 节简单介绍了命令中的管道，重定向输入/输出和错误。管道及重定向功能在实际应用中非常频繁，因此初学者应该理解并掌握这些内容。

本章主要介绍了 Linux 系统的入门知识、基本操作等内容，有了这些基础知识后，从第 3 章起将开始讲解如何管理和使用 Linux 系统。

第 3 章　常 用 命 令

与 Windows 系统不同，即使 Linux 处于图形环境中，仍然需要使用大量的命令（例如关闭、重启系统服务等），因此命令在 Linux 系统中十分重要。

本章主要涉及的知识点如下。

- ❑ Linux 系统中的基本命令的使用方法介绍。
- ❑ 常见的文件操作命令及其使用方法。
- ❑ 查看、修改系统的日期和时间。
- ❑ 查看和管理联线用户命令介绍。
- ❑ 切换用户命令的使用。
- ❑ 定位、查找文件和关键字命令的使用。
- ❑ 常见的输入/输出命令。

3.1　Linux 基本命令

Linux 系统中包含了几个使用频率非常高的命令，例如切换路径、查看当前所在路径、查看目录中的文件列表等，通常将这些命令称为基本命令。本节将简单介绍 Linux 中的基本命令及使用方法。

3.1.1　切换工作目录命令 cd

当用户从字符界面或相关终端上登录并工作时，需要频繁地切换当前工作目录，这时就需要使用到切换工作目录命令 cd。本小节将简单介绍切换工作目录命令 cd 及其使用方法。

【命令格式】

```
cd [dir]
```

【参数说明】

参数 dir 表示要切换目录的路径，路径的表示方法可以是相对路径，也可以是绝对路径。如果没有使用任何参数，cd 命令将切换工作目录至用户的家目录。

【路径表示方法】

在 Linux 系统中，表示路径的方法有两种，二者的不同之处在于路径的起始位置不同：第 1 种表示方法以根文件系统（即/）为起点，这种方法被称为绝对路径。例如/root、/home、/var/log 等以/开头的都是绝对路径。第 2 种表示方法以当前目录为起点，这种方法称为相对路径。相对路径使用点号“.”表示当前目录，两个点号“..”表示当前目录的上一级目

录。例如./test、./test/a1、../etc/samba/等，以“.”和“../”开头的都属于相对路径。

【用法示例】

（1）切换路径

使用切换路径命令 cd 将当前工作目录切换到/etc：

```
#进入当前目录的上级目录
[user1@localhost ~]$ cd ../
#进入当前目录上级目录的 etc 目录
[user1@localhost home]$ cd ../etc
[user1@localhost etc]$
```

将当前工作目录切换到系统日志文件目录/var/log：

```
#进入/var/log 目录
[user1@localhost ~]$ cd /var/log
[user1@localhost log]$
```

从命令提示符的变化可以看出当前的工作目录发生了变化。

（2）快速返回家目录

由于符号“~”表示当前用户的家目录，因此可以使用以下命令快速回到用户的家目录：

```
#进入家目录
[user1@localhost log]$ cd ~
[user1@localhost ~]$
```

cd 命令将改变当前的工作目录为参数所指向的目录，如果没有使用任何参数，命令也会将当前工作目录切换到用户的家目录中：

```
#当 cd 命令的参数为空时，cd 命令会将当前工作目录切换到用户的家目录中
[user1@localhost log]$ cd
[user1@localhost ~]$
```

注意：与 Windows 不同，在 Linux 系统中文件、目录名称、命令字等均对大小写敏感，因此在使用文件及命名文件时都应该特别注意。

3.1.2　查看当前路径命令 pwd

在 Linux 系统中可能会有许多名称相同的目录，有时进入一个目录后可能无法判断当前工作目录的具体位置。Linux 系统提供了一个用于查看当前目录的命令 pwd，该命令用于输出当前工作目录的完整路径（即绝对路径）。

【命令格式】

```
pwd
```

pwd 命令没有任何选项和参数。

【用法示例】

输出当前目录的完整路径：

```
[user1@localhost log]$ pwd
/var/log
```

使用相对路径切换工作目录并查看完整路径的例子：

```
[user1@localhost ~]$ pwd
/home/user1
[user1@localhost ~]$ cd ../../etc
#切换当前工作目录后使用 pwd 命令输出目录完整路径
[user1@localhost etc]$ pwd
/etc
```

3.1.3　查看文件列表命令 ls

许多时候需要查看某个目录中的文件列表，以便于对其中的某个文件进行操作，这时可以使用 Linux 提供的 ls（可速记为 list）命令。ls 命令不仅可以查看目录中包含的文件列表，还可以查看文件的详细信息，例如查看文件的权限、大小、所有者等。本小节将简单介绍 ls 命令及其常见用法。

【命令格式】

```
ls [option [filename]]
```

【常用选项】

- ❑ l：以长格式显示文件列表。
- ❑ d：显示目录的详细信息，而非目录中的文件列表。
- ❑ a：显示隐藏文件。
- ❑ k：显示文件大小时以 k 字节为单位。
- ❑ S：按文件大小进行排序。
- ❑ h：以更直观的方式查看文件列表（显示的文件大小信息更加直观），这个选项必须配合选项 l 使用。

【参数说明】

ls 命令可以使用的参数有以下两类。

- ❑ 文件：命令将会返回参数指定的文件。例如使用 l 选项时，将返回指定文件的详细信息。
- ❑ 目录：若配合使用 d 选项，则返回目录本身，否则将返回目录中的文件列表。

如果没有指定要查看的目录，命令将返回当前工作目录中的文件列表。

【用法示例】

（1）要查看当前目录下的文件列表：

```
#当 ls 命令参数为空时，将列出当前目录中的文件列表
# ls
account  crash  empty   gdm local   log    nis preserve   run    tmp yp
cache    db     games   lib lock    mail   opt raccoon    spool  www
```

（2）许多时候需要查看文件及目录的详细信息，这时可以加上选项 l。例如要查看当前目录中的文件详细信息：

```
#以长格式的形式显示当前目录中的文件列表
# ls -l
total 68
-rw-r--r--  1 root root   17 Sep  7 05:11 123
-rw-------  1 root root 1332 Sep  3 07:16 anaconda-ks.cfg
drwxr-xr-x  2 root root 4096 Sep  4 08:14 Desktop
……
```

上面这条命令输出目录中的文件详细信息，使用空格可以将详细信息分为多个字段，各字段的含义如下。

- 第 1 个字段的第 1 个字符描述了文件的类型，上面的输出中有“-”和“d”两种文件类型，其后的 9 个字符代表权限（将在文件权限相关章节中介绍）。
- 文件的连接数。
- 文件的属主和属组。
- 文件的大小。
- 文件创建的时间。
- 文件及目录名。

由于选项 l 可以查看到文件的详细信息，因此 ls 命令的这个用法最为常见，初学者应该掌握这个用法。除此之外，RHEL5.3 还为命令 ls -l 定义了别名 ll，用户可以使用 ll 这个快捷命令代替 ls -l。

在 ls 命令的长格式输出中，使用第 1 个字段的第 1 个字符表示文件的类型，常见的类型及其对应的字符如下。

- –：减号表示普通文件。
- d：字母 d 表示一个目录。
- l：表示链接文件，类似于 Windows 下的快捷方式。
- b：块设备文件，通常是磁盘或分区。
- c：字符设备文件，通常是键盘、鼠标和终端等。
- p：管道文件。

小知道： 在 RHEL 等操作系统中，ls 命令使用不同的颜色标识不同的文件类型，例如使用绿色表示可执行文件、蓝色表示目录等，可以使用此方法快速确定文件的类型。

（3）可以指定查看一个文件，例如要查看文件/dev/sdb1：

```
#使用 ls 命令查看一个块设备文件
# ls -l /dev/sdb1
brw-r----- 1 root disk 8, 17 08-27 10:14 /dev/sdb1
```

（4）当命令 ls 的参数为一个目录时，命令将会输出目录中所有的文件列表。例如要查看目录/usr 中的文件：

```
#查看目录/usr 中的文件列表
# ls -l /usr
total 188
```

```
drwxr-xr-x   2 root root    45056   Sep 4   06:59 bin
drwxr-xr-x   2 root root    4096        Aug 8   2008 etc
drwxr-xr-x   2 root root    4096        Aug 8   2008 games
……
```

（5）要查看目录本身而不是其中的文件，应该使用选项 d。例如要查看目录/usr：

```
#使用 d 选项查看目录本身
# ls -ld /usr
drwxr-xr-x 14 root root 4096 Sep  3 06:53 /usr
```

（6）Linux 系统中的隐藏文件都使用点号“.”开头，默认情况下，ls 命令不会显示这些隐藏文件，要查看隐藏文件，应该配合使用选项 a。

例如查看当前目录中的所有文件：

```
#使用选项 a 查看当前目录中的隐藏文件
# ls -al
total 220
drwx------  17  user1   500     4096    Sep  7 06:44 .
drwxr-xr-x  4   root    root    4096    Sep  4 06:01 ..
-rw-------  1   user1   user1   242     Sep  7 06:53 .bash history
……
```

从上面的输出中可以看到相对路径“.”和“..”都被认为是隐藏文件。

（7）ls 命令使用字节为单位表示文件大小，这样的输出看起来很不方便。此时可以使用选项 k 以 k（千）字节为单位查看文件大小，例如：

```
#查看当前目录中的文件列表时，以千字节为单位显示文件大小
# ls -akl
total 220
drwx------  17  user1   500     4 Sep  7 06:44 .
drwxr-xr-x  4   root    root    4 Sep  4 06:01 ..
-rw-------  1   user1   user1   1 Sep  7 06:53 .bash history
……
```

（8）使用选项 k 后，虽然看起来比较方便，但一些体积较大的文件看起来仍然很困难，这时可以使用选项 h 以比较直观的方式查看文件大小。

例如以比较直观的方式查看当前目录中的文件：

```
#以更直观的方式（合适的单位）显示当前目录中的文件列表
# ls -lh
total 9.0M
drwxr-xr-x      2   admin admin 4.0K   Jul  5 13:23 Deployment
drwxr-xr-x      14  admin admin 4.0K   Jul  5 13:21 extensions
drwxr-xr-x      54  admin admin 4.0K   Jul  5 13:12 payloads
drwxr-xr-x      2   admin admin 4.0K   Jul  5 13:09 redist
drwxr-xr-x      5   admin admin 4.0K   Jul  5 13:09 resources
-rwxr--r--      1   admin admin 2.9M   Jan 28 2010 Setup.exe
……
```

从命令输出中可以看出 h 选项为大小不同的文件使用了不同的单位，这样看起来就比较方便了。

（9）使用选项 S 可以按文件大小进行排序，例如：

```
#显示文件列表的同时按文件大小排序
# ls -lhS
total 9.0M
-rwxr--r--  1   admin admin 3.8M   Sep 20  2008 WinBootstrapper.msi
-rwxr--r--  1   admin admin 2.9M   Jan 28  2010 Setup.exe
-rwxr--r--  1   admin admin 2.4M   Sep 19  2008 WinBootstrapper1.cab
drwxr-xr-x  2   admin admin 4.0K   Jul 5   13:23 Deployment
……
```

3.1.4　文件链接命令 ln

文件链接类似于 Windows 下的快捷方式，它是指向一个文件的链接。Linux 系统中创建文件链接使用命令 ln（可速记为 link），其创建的链接可以分为硬链接和软链接。

【命令格式】

```
ln [OPTION] Target Link_name
```

【常用选项】

ln 命令最常用的选项为 s，使用该选项之后创建的将是软链接。

【参数说明】

ln 命令使用的参数有如下两个。

- Target：目标文件。
- Link_name：创建后的链接文件名称。

【软、硬链接文件的区别】

软、硬链接文件的区别在于：软链接文件是一个指向目标文件的快捷方式（指向的目标文件也可以不存在），这与 Windows 系统中的快捷方式的概念完全相同。而硬链接文件则是目标文件的“副本”，这个“副本”与普通文件非常相似，不同的是“副本”在磁盘上的存储位置与目标文件一致。即无论目标文件存在多少个“副本”，这些“副本”都使用同一块存储区域，当其中一个“副本”的内容发生变化时，所有“副本”和目标文件的内容都会发生变化。

【用法示例】

（1）创建硬链接

不使用任何选项时 ln 将会创建一个硬链接文件：

```
#为文件/home/user1/a 建立一个硬链接文件，并命名为/home/user1/b
$ ln /home/user1/a /home/user1/test/b
```

查看建立的硬链接文件：

```
$ ls -l /home/user1/test/b
#硬链接文件与普通文件显示相同
-rw-rw-r-- 2 user1 user1 10 Sep  8 04:29 /home/user1/test/b
```

（2）创建软链接

创建一个软链接需要使用选项 s，例如：

```
#使用 s 选项建立软链接文件
$ ln -s /home/user1/a /home/user1/test1/c
```

查看建立的软链接文件：

```
$ ls -l /home/user1/test1/c
#查看软链接文件时，将显示链接的源文件信息
lrwxrwxrwx 1 user1 user1 13 Sep  8 04:32 /home/user1/test1/c -> /home/user1/a
```

从上面的命令也可以看出，硬链接是目标文件的一个“副本”，而软链接则建立一个指向目标文件的镜像。

注意：在使用文件链接时，硬链接不能跨越文件系统（即分区），而软链接可以跨越文件系统。

3.2　文件操作命令

无论何种操作系统，在日常使用过程中，文件操作都是一项最基本的工作，涉及的文件操作主要有复制、移动、删除文件等。本节将简单介绍 Linux 系统中文件操作的相关命令及其使用方法。

3.2.1　文件命名规则

Linux 系统的命名规则与 Windows 系统相差较大，因此在学习文件操作之前，应该对 Linux 系统的文件命名和使用规则有一定的了解。

（1）文件命名规则

Linux 系统中的文件名称最长可以有 256 个字符，文件名称可以由大小写字母、数字、下划线、减号及一些特殊符号组成。这些特殊符号可以是空格、$（美元符号）、?（问号）、*（星号）等，这些特殊的符号通常都具有特殊的含义（特殊符号将在第 4 章中介绍），因此在创建文件时通常都不推荐使用这些符号。除此之外，由于斜杠“/”是路径分隔符，因此也不能在文件名中使用。

（2）文件扩展名

Linux 系统中没有文件扩展名的概念，因此在为文件命名时，不需要强制使用文件扩展名。虽然如此，有时也使用一些特殊的文件扩展后缀表示文件的用途，例如通常使用“.conf”结尾表示一个配置文件，“.sh”表示一段 Shell 程序，“.log”表示日志文件等。

虽然 Linux 中的文件没有扩展名的概念，但某些特殊的程序要求文件必须有正确的扩展名才能使用（例如 gcc 编译器要求 C 语言的源码文件必须使用“.c”结尾），因此应该注意阅读相关程序的说明。

（3）特殊字符和隐藏文件

为一个文件命名时，为避免系统误解文件名，通常不建议使用特殊字符作为文件名称的一部分，例如使用一个包含空格的文件名，在查看或创建时容易被误解为多个文件。如果确实需要使用特殊字符作为文件名的一部分，可以使用反斜杠“\”屏蔽这些字符的特殊含义。

在 Linux 系统中，以“.”点号开头的文件都是隐藏文件，因此除非确实需要，否则不要使用“.”点号作为文件名的开头。

注意：在 Linux 系统中，同一目录下不允许文件和目录同名，即在同一目录下不允许同时存在名称相同的文件和目录。

3.2.2　创建文件命令 touch

touch 命令是一个比较常用的文件，它可以用来创建一个空文件，也可以用来修改文件的创建时间。

【命令格式】

```
touch [option] filename
```

【命令参数】

touch 命令会创建以参数 filename 为名称的文件，因此参数 filename 应该遵循文件命名规则。

【用法示例】

（1）创建空文件

要创建一个文件 ab，可以使用命令：

```
#使用 touch 命令创建一个空文件
# touch ab
# ls -l ab
-rw-r--r-- 1 root root 0 Sep 11 03:08 ab
```

由于新创建的文件没有任何内容，因此使用 touch 命令创建的文件都是空文件。

（2）创建并修改文件的时间戳记

要创建一个新文件并修改其时间戳记，可以使用命令：

```
#创建文件时，使用选项 d 指定时间戳记
# touch -d "6/20/10 18:32" ed
# ls -l ed
-rw-r--r-- 1 root root 0 Jun 20 18:32 ed
```

从上面的命令可以看出，touch 成功地创建并修改了文件的时间戳记。

3.2.3　创建目录命令 mkdir

对目录的操作中，最常见的任务是创建一个目录，在 Linux 中创建目录使用的命令是 mkdir。

【命令格式】

```
mkdir [option] dir
```

【命令选项】

mkdir 命令最常用的选项是 p，它的功能是可以同时创建一个路径中的多个目录。

【用法示例】

（1）例如要创建一个名为 test 的目录：

```
#使用 mkdir 命令创建目录并查看
# mkdir test
# ls -ld test
drwxr-xr-x 2 root root 4096 Sep 11 04:06 test
```

（2）有时需要按路径创建目录，如果依次创建目录将会非常麻烦，这时可以配合选项 p。例如要在当前目录下创建多个目录，其位置关系为“./a/b/c/d/e”：

```
#按路径创建目录
# mkdir -p a/b/c/d/e
#进入并查看创建的目录
# cd a/b/c/d/e/
# pwd
/home/user1/a/b/c/d/e
```

3.2.4　移动、重命名文件命令 mv

在 Linux 系统中，移动、重命名文件可以使用命令 mv（可速记为 move）。

【命令格式】

```
mv Sou_file dir
```

【参数说明】

使用 mv 命令时需要指定如下两个参数。

- Sou_file：需要移动的文件或目录名。
- dir：移动后的位置和文件名。如果此参数指定的文件已经存在，则 mv 将覆盖已存在的文件；若不存在，则移动文件并重命名。

【用法示例】

（1）要将当前目录中的文件 a 移动到目录 test 中，可以使用如下命令：

```
#将当前目录中的文件 a 移动到目录 test 中
$ mv a test/
$ ls -l test
total 16
-rw-r--r-- 1 user1 user1 9733 Sep 11 05:01 a
```

（2）mv 也可以用来移动一个目录。例如将目录 test 移动到目录 file 中：

```
#将目录 test 移动到目录 file 中
$ mv test file/
$ ls -l file
total 8
drwxrwxr-x 2 user1 user1 4096 Sep 11 05:01 test
```

（3）mv 一次可以移动多个文件。例如将文件 b、c、d 移动到目录 file 中：

```
#将 b、c、d 移动到目录 file 中
$ mv b c d file/
```

```
$ ls -l file
total 20
-rw-rw-r--      1 user1 user1    0 Sep 11 05:26 b
-rw-rw-r--      1 user1 user1    0 Sep 11 05:26 c
-rw-rw-r--      1 user1 user1    0 Sep 11 05:26 d
drwxrwxr-x  2 user1 user1 4096 Sep 11 05:01 test
```

（4）mv 命令还可以用来重命名文件或目录。例如将目录 file 重命名为 test：

```
#使用 mv 命令将目录 file 重命名为 test
$ mv file test
$ ls
test
```

3.2.5　复制文件命令 cp

许多时候需要备份一个简单的文件，例如备份一个配置文件，以便于需要时能够恢复文件的内容，这时就需要复制文件并执行备份。Linux 系统中复制文件的命令是 cp（可速记为 copy）。

【命令格式】

```
cp [option] Source Directory
```

【常用选项】

cp 命令中有一个常用选项 R，它可以递归的复制目录中的所有文件。

【参数说明】

cp 命令有如下两个参数。

- Source：要复制的源文件。
- Directory：复制文件的新位置。如果此参数是一个新目录名，则将文件复制到新位置时重命名文件。

【用法示例】

（1）复制并重命名文件

将/etc/samba/smb.conf 备份到当前目录中，并将文件名重命名为 smb.conf_backup：

```
#备份 Samba 服务的配置文件并查看其权限变化情况
$ cp /etc/samba/smb.conf smb.conf_backup
#查看并对比复制后的文件权限变化
$ ls -l /etc/samba/smb.conf
-rw-r--r-- 1 root root 9733 Dec 10  2008 /etc/samba/smb.conf
$ ls -l smb.conf_backup
-rw-r--r-- 1 user1 user1 9733 Sep 11 06:28 smb.conf_backup
```

从上面命令的结果中可以看出，复制后的文件属主和属组虽然发生了变化，但文件的权限却没有发生变化。

（2）制作光盘镜像

cp 命令还有一个相当实用的功能，即用来制作光盘镜像。例如将当前光驱中的光盘制作成光盘镜像文件：

```
#使用 cp 命令为当前光驱中的光盘制作镜像
# cp /dev/cdrom linux.iso
#将制作的光盘镜像挂载到/media 目录
# mount -o loop linux.iso /media
```

上面的命令先将光驱设备中的光盘制作成镜像文件，然后使用 mount 命令将镜像挂载到/mnt 目录（挂载命令将在第 7 章中介绍）。

（3）复制目录

如果要复制的源目录中还存在子目录，可能会发生错误，此时可以使用选项 R 递归地复制子目录。例如要将目录 test 复制到目录 file 下：

```
#使用 R 选项递归地复制目录及目录中的文件
$ cp -R test file/
```

（4）复制时保留权限

在对文件进行复制的过程中，复制的新文件通常都具备了新的文件属主、属组和权限信息，有时可能需要保存原有文件的权限等信息（例如防止复制的备份文件被修改等）。这时可以使用 p 选项。例如要备份防火墙配置文件：

```
#使用选项 p 保持复制的文件权限
# cp -p /etc/sysconfig/iptables ./backup
```

p 选项虽然不是一个常用的选项，但在对文件进行复制备份时，可以让用户不必为复杂的权限而担心，因此在备份一些需要保护权限的文件时，建议使用这个选项。

3.2.6　删除文件命令 rm

有时保存的文件会失去存在的意义（即以后都不会再使用这个文件），此时为了释放存储空间，就需要删除这些无用的文件，删除文件可以使用命令 rm（可速记为 remove）。

提示：使用 rm 命令删除的文件将会直接从硬盘中删除（Linux 系统不存在回收站的概念），因此删除文件时应该特别谨慎。

【命令格式】

```
rm [option] file
```

【常用选项】

rm 命令有如下两个常用的选项。

- r：删除目录时将目录中的所有内容一并删除。
- f：忽略删除的目录中不存在的子目录，并且删除时不提示用户。

【用法示例】

（1）例如要删除文件 ab：

```
$ rm ab
#删除文件 ab 时系统会要求用户确认
rm: remove regular file `ab'? y
```

rm 会提示用户是否确实需要删除文件，确认之后即可删除文件。

（2）通常删除目录及目录中的所有文件和子目录时，常见的用法是配合使用选项 r 和 f。例如删除目录 test 并删除其中的所有文件：

```
#删除目录 test 及其中的文件并且不要求用户确认
$ rm -rf test
```

使用此方法删除目录时，用户不会得到任何提示，因此应该先确认目录中的所有文件都需要删除。

（3）删除目录中的所有文件时，可以使用“*”表示所有文件。例如删除当前目录中的所有文件及目录：

```
#删除当前目录中的所有文件及目录
$ rm -rf *
```

3.2.7　删除空目录命令 rmdir

Linux 系统中提供了一个专用于删除目录的命令 rmdir，但由于此命令功能有限（只能删除空目录），因此较少使用。

【命令格式】

```
rmdir [option] directory
```

mkdir 命令没有过多的选项和参数，使用时直接指定需要删除的目录即可。

【用法示例】

（1）删除空目录 file：

```
#删除一个空目录 file
$ rmdir file
```

（2）同 mkdir 命令一样，rmdir 命令也提供了一个选项 p，该选项主要用于删除一个路径上的目录。例如删除空目录 a/b/c/d：

```
#使用 p 选项按路径删除空目录
$ rmdir -p a/b/c/d/
```

rmdir 命令最常见的用法是同 find 命令（将在第 5 章中介绍）配合，删除某个目录中的空目录。

提示：由于 rmdir 仅能删除空目录，因此大多数时候都使用 rm 命令删除目录。

3.2.8　查看文件类型命令 file

由于 Linux 系统中的文件没有使用扩展名，因此需要借助命令来查看文件的类型，查看文件类型可以使用命令 file。

【命令格式】

```
file filename
```

使用 file 命令时，只需要指定要查看的文件即可。

【用法示例】

（1）查看文件 a 的文件类型：

```
$ file a
#以下为文本文件的提示信息
a: ASCII English text
```

由上面的结果可以看出文件 a 是一个文本文件。

（2）查看文件/usr/bin/file 的类型：

```
#查看文件/usr/bin/file 的文件类型
$ file /usr/bin/file
#以下为可执行文件的提示信息
/usr/bin/file: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
for GNU/Linux 2.6.9, dynamically linked (uses shared libs), for GNU/Linux
2.6.9, stripped
```

从命令的输出中可以看出，这是一个可执行文件。

（3）查看建立的链接文件：

```
$ file b
#以下为链接文件的提示信息
b: broken symbolic link to `b'
```

命令的结果仅说明文件 b 是一个链接文件。

有时可能希望命令输出链接指向的文件类型，这时应该使用选项 L。追溯到源文件查看其类型：

```
$ file -L /bin/csh
#使用选项 L 追溯源文件查看文件类型
/bin/csh: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for
GNU/Linux 2.6.9, dynamically linked (uses shared libs), for GNU/Linux 2.6.9,
stripped
```

使用选项 L 时，链接文件的层数不能太深，否则可能会出现错误。

3.3　文本文件内容相关命令

在 Linux 系统中，许多系统及服务设置都是保存在文本文件中的，有时为了查看一些设置（例如查看网络接口的配置文件），不得不查看相关文本文件的内容。为此 Linux 系统中提供了几个功能各异的查看文本内容的命令，本节将简单介绍这些命令及其使用方法。

3.3.1　查看文本文件内容命令 cat

cat 命令的作用是查看文本文件内容，使用率极高。它将文本内容输出到屏幕上，然后返回 Shell，从其功能可以看出 cat 命令非常适合查看行数较少的文本。

【命令格式】

```
cat [option] filename
```

【常用选项】

cat 命令的常用选项只有 v，其功能是显示文件内容的同时，也显示文件中的控制字符，这个选项非常方便查看脚本中不能识别的控制字符。

【用法示例】

（1）查看文本 smb.conf 的内容：

```
#使用 cat 命令查看文本文件 smb.conf 的内容
# cat smb.conf
[global]
        workgroup = MYGROUP
        server string = Samba Server Version %v
        security = user
……
```

（2）有些用户可能会利用一些熟悉的编辑工具完成脚本编写（例如 UltraEdit 文本编辑器），然后利用工具将其上传到 Linux 系统中。然而这些文本文件可能会存在 Linux 系统无法理解的一些控制字符，可以使用选项 v 查看这些不可见的控制字符。例如查看文件 test.sh 的内容并显示其中的控制字符：

```
#查看文件 test.sh 内容的同时显示其中的控制字符
$ cat -v test.sh
#!/bin/bash^M
echo "This is a test script"^M
exit 0^M
```

其中的“^M”就是控制字符，这个字符就是 Windows 系统文档中的换行符。

提示： 如果初学者找不到用于练习的文本文件，可以复制一个系统的配置文件（例如 /etc/samba/smb.conf 等）进行练习。

3.3.2　从文本尾查看文本内容命令 tail

tail 命令的作用是从文本最后一行开始查看文本，这是一个非常有用的命令，管理员经常使用这个命令查看新产生的日志消息（新日志消息通常在日志文件的最后）。

【命令格式】

```
tail [option] filename
```

【常用选项】

- n：指定查看的行数。
- f：动态地显示文件内容的变化情况。

【用法示例】

（1）要查看系统日志文件的最后几行：

```
#使用 tail 命令从文件结尾开始查看内容
#默认情况下显示文本最后 10 行
# tail /var/log/messages
Sep 11 08:34:01 localhost smartd[3220]: Device: /dev/sdd, IE (SMART) not
enabled, skip device Try 'smartctl -s on /dev/sdd' to turn on SMART features
Sep 11 08:34:01 localhost smartd[3220]: Monitoring 0 ATA and 0 SCSI devices
…………
```

（2）tail 默认查看文件的最后 10 行，如果要自定义行数，可以使用选项 n。例如要查看系统日志文件的最后 5 行：

```
#使用选项-5 指定查看文件的最后 5 行
# tail -5 /var/log/messages
Sep 11 18:17:03 localhost restorecond: Will not restore a file with more
than one hard link (/etc/resolv.conf) Invalid argument
Sep 12 00:41:52 localhost rz[5276]: [root] test.sh/ZMODEM: 49 Bytes, 5162
BPS
Sep 12 00:48:18 localhost rz[5324]: [user1] test.sh/ZMODEM: 51 Bytes, 789
BPS
……
```

（3）tail 命令还有一个很特殊的选项 f，这个选项可以动态地显示文本内容的变化。例如监控系统日志文件 messages：

```
#使用 f 选项跟踪文件内容的变化情况
# tail -f /var/log/messages
```

tail 显示完文件的最后几行之后并没有返回，而是一直监控其内容的变化，如果文件最后被追加新的内容，tail 将会立即更新显示。如果需要退出监控状态，可以使用快捷键 Ctrl+C 强制退出（这个快捷键对许多命令和程序都起作用）。

3.3.3　从文本首行查看文本内容命令 head

head 命令与 tail 正好相反，head 将从首行开始显示文本的内容，这在查看长文件的前几行时非常有用。

【命令格式】

```
head [option] filename
```

【常用选项】

head 命令的常用选项只有一个 n，其功能是指定查看的行数。

【用法示例】

（1）使用 head 命令查看用户信息文件/etc/passwd 的前几行：

```
#使用 head 命令查看文本的前几行
#默认显示前 10 行
# head /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

```
adm:x:3:4:adm:/var/adm:/sbin/nologin
…………
```

（2）与 tail 一样，head 默认也只显示文本的前 10 行，如果要指定显示的行数，可以使用选项 n。例如查看用户信息文件的前 3 行：

```
#使用选项-3 指定显示文本的前 3 行
# head -3 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

3.3.4　分屏显示文本内容命令 more 和 less

前面几条命令都显示文件的一部分，或将文本所有内容输出到屏幕并返回，这在查看一些很长的文本时不太方便。为此 Linux 提供了两个交互式的文本查看命令：more 和 less。本小节将简单介绍这两个命令及其使用方法。

1. more命令

more 是一个交互式的文本查看工具，经常用于查看内容较长的文本文件。

【命令格式】

```
more [option] filename
```

使用 more 命令时，通常都不需要使用选项，直接指定要打开的文件即可。

【用法示例】

（1）使用 more 命令查看文件 smb.conf 的内容：

```
#使用 more 命令查看文本 smb.conf 的内容
$ more smb.conf
#以下为交互式页面的内容
# This is the main Samba configuration file. You should read the
# smb.conf(5) manual page in order to understand the options listed
# here. Samba has a huge number of configurable options (perhaps too
# many!) most of which are not shown in this example
……
#
# NOTE: Whenever you modify this file you should run the command "testparm"
# to check that you have not made any basic syntactic errors.
#
#---------------
# SELINUX NOTES:
--More--(10%)
```

屏幕最下方将会显示已查看的内容占整个文本的百分比，其常用的几个交互命令如下。

- Enter 键：显示下一行内容。
- 空格键：显示下一屏内容。
- b：显示上一屏的内容。
- q：退出交互界面并返回。

其他一些交互命令，可以按 h 键查看帮助：

```
#在交互式页面中按 h 键查看帮助页面
Most commands optionally preceded by integer argument k.  Defaults in
brackets.
Star (*) indicates argument becomes new default.
----------------------------------------------------------------------
<space>                 Display next k lines of text [current screen size]
z                       Display next k lines of text [current screen size]*
<return>                Display next k lines of text [1]*
………………
----------------------------------------------------------------------
```

如果用户需要从帮助页面中返回，可以按 Enter 键或空格键，more 命令将重新显示文本的内容。当用户使用 Enter 键或空格键将屏幕滚动到文本末尾时，more 将会自动退出并返回命令提示符。

（2）more 命令的另一个重要用途是查看较长的命令输出，例如：

```
#输出系统安装的所有软件包并交给 more 显示
# rpm -aq | more
```

2. less命令

与 more 命令一样，less 也采用分屏式的方法查看文本，不同的是 less 在其基础上扩展了一些功能，操作更加便捷。

【命令格式】

```
less [option] filename
```

less 命令没有常用的选项，通常只需要为其指定要打开的文件名即可。

【用法示例】

（1）使用 less 命令查看文件 smb.conf：

```
#使用 less 命令查看 smb.conf 的内容
$ less smb.conf
#以下是交互式页面内容
# This is the main Samba configuration file. You should read the
……
# NOTE: Whenever you modify this file you should run the command "testparm"
# to check that you have not made any basic syntactic errors.
#
#---------------
# SELINUX NOTES:
smb.conf
```

从上面的 less 界面中可以看出，使用 less 命令查看文本时，底部将显示打开的文本名称（more 命令显示文本的百分比）。

此时用户可以使用以下交互指令查看文本的内容。

- ❑ Page Up/Page Down：跳转到上一页、下一页。
- ❑ Home/End：跳转到第一页、最后一页。

- ❑ 上、下方向键：向上、向下滚动一行。
- ❑ /pattern：向后查找字符串 pattern 并高亮显示这些字符串。使用 n、N 键跳转到下一个、上一个匹配字符串。
- ❑ ?pattern：向前查找字符串 pattern 并高亮显示找到的字符串。
- ❑ g/G 键：跳转到文件第一行、最后一行。
- ❑ q：退出 less。

（2）如果需要获取 less 的帮助，可以按 h 键：

```
#按 h 键将显示以下帮助界面
                    SUMMARY OF LESS COMMANDS

      Commands marked with * may be preceded by a number, N.
      Notes in parentheses indicate the behavior if N is given.

  h  H               Display this help.
  q  :q  Q  :Q  ZZ    Exit.
 ---------------------------------------------------------------------------

                            MOVING

  e  ^E  j  ^N  CR  *  Forward  one line   (or N lines).
……
HELP -- Press RETURN for more, or q when done
```

与 more 不同的是跳转到文件结尾处时，less 不会返回到提示符，而是等待用户继续输入命令，这样就很方便用户前后翻页并查看。

3.3.5　文本内容比较命令 diff

有时需要比较两个文件的内容（例如制作补丁包时），这时可以使用系统自带的 diff 命令。

【命令格式】

```
diff [optio] file1 file2
```

【常用选项】

- ❑ c：以上下文的形式显示两个文件的不同之处，通常是 3 行。
- ❑ C：以上下文的形式显示两个文件的不同之处，上下文的行数由选项指定的参数决定。
- ❑ b：忽略空格字符的不同。
- ❑ B：忽略空白行的不同。
- ❑ u：以合并的方式显示比较的结果。
- ❑ r：递归地比较两个目录及子目录中的文件，此时的参数应该使用两个目录。

diff 命令可以使用的选项非常多，但许多都不常用，此处仅简单介绍几个较为常用的选项。

【用法示例】

为了讲解命令的使用，此处引入两个示例文档 file1 和 file2，查看这两个文档的内容

如下：

```
#使用 cat 命令查看示例文档 file1 的内容
# cat file1
a
b
c
d
e
#使用 cat 命令查看示例文档 file2 的内容
# cat file2
b
c
d
g
h
```

（1）可以不加任何选项使用 diff 命令对文件进行比较：

```
#使用 diff 命令比较示例文档 file1 和 file2 的内容
# diff file1 file2
1d0
< a
5c4,5
< e
---
> g
> h
```

这个输出结果看起来让人很费解，下面简单介绍其输出的格式。输出结果最前面的“1d0”表示第 1 个文件比第 2 个文件多一行，而其后的“<”表示第一个文件的不同之处，“>”表示第 2 个文件的不同之处。3 个减号“---”表示两个文件中被省略的相同部分，输出结果中的“5c4,5”表示第 1 个文件的第 5 行和第 2 个文件的第 4、5 行内容不同。

（2）也可以使用上下文的方式对文档进行比较，例如要对示例文档进行上下文方式的比较，并设置上下文行数为 1：

```
#设置上下文 1 行比较文档内容
# diff -C 1 file1 file2
*** file1      2011-01-01 05:30:01.000000000 -0800
--- file2      2011-01-01 05:30:36.000000000 -0800
***************
*** 1,2 ****
- a
  b
--- 1 ----
***************
*** 4,5 ****
  d
! e
--- 3,5 ----
  d
! g
! h
```

相比之前的输出，此时的输出要直观许多。

也可以配合使用其他选项进行更复杂的文档比较操作，diff 命令虽然不是一个常用的命令，但在许多时候却非常有用（例如制作一个升级文件、比较备份文件与当前文件有何不同）。

3.3.6　文本统计命令 wc

许多时候需要对一个文档或命令的输入做一个统计，例如要统计命令的输出一共有多少行，或者命令输出、文档的字数统计等，这时可以使用 wc 命令来完成这些工作。

【命令格式】

```
wc [option] file
```

【常用选项】

- ❑ c：统计文件的字节数并输出。
- ❑ m：统计并显示文件的字符数。
- ❑ l：统计并显示文件的行数。
- ❑ L：显示文件中最长行的长度。
- ❑ w：统计并输出文件的字数。

【用法示例】

（1）可以不加任何选项直接使用 wc 命令对一个文件进行统计：

```
#使用 wc 命令统计 a.txt 的内容
# wc a.txt
 288 1672 9733 a.txt
```

命令依次输出了文件 a.txt 的行数、词数、字节数及文件名称。

（2）许多时候可能仅需要统计其中一项，这时就可以配合相应的选项进行统计。例如要统计文件 a.txt 的字数：

```
#使用 w 选项统计文件的字数
# wc -w a.txt
1672 a.txt
```

统计文件 a.txt 的行数：

```
#使用 l 选项统计文件的行数
# wc -l a.txt
288 a.txt
```

关于 wc 命令的其他用法，读者可以参考上面的示例进行一一验证。

3.4　日期时间命令

对于某些服务器而言（例如依赖于定时任务的服务器），时间的准确性十分重要，因此管理员的一项重要工作是查看和校准系统时钟。本小节将简单介绍如何查看和修改

Linux 系统中的日期和时间。

3.4.1　查看日期时间命令 date

目前大多数 UNIX 和 Linux 系统，都使用自 1970 年 1 月 1 日至今的秒数计算并保存时间，因此 Linux 系统中的时间计算多使用秒数。在 Linux 系统中如果需要查看日期和时间，可以使用 date 命令。

【命令格式】

```
date [option] [+FORMAT]
```

【常用选项】

date 命令常用的选项只有一个 s，其作用是从指定的字符串中读取并设置当前系统的时间。

【用法示例】

（1）使用“date”命令查看当前系统时间：

```
#使用 date 命令显示当前系统的时间
$ date
Sat Sep 11 23:59:31 CST 2010
```

（2）date 命令还提供了许多用于格式化输出时间的选项（即 FORMAT 格式化字符串），可以利用这些选项自定义输出的时间格式。

例如按 year-month-day　HH:MM:SS 的格式输出时间：

```
#使用 date 命令输出时间的同时格式化输出
$ date +"%F  %T"
2010-09-12  00:00:16
```

在上面这条命令中，date 命令附带的参数就是一个格式化字符串（格式化字符串必须放入双引号内）。除此之外，date 命令还可以使用许多格式化选项，读者可以参阅 date 命令的帮助信息了解更多的格式化选项。

3.4.2　查看日历命令 cal

用户可能希望像查看日历一样查看当前的日期，在 Linux 系统中提供了一个用于查看日历的命令 cal。

【命令格式】

```
cal [option] [[month] year]
```

【常用选项】

cal 命令常用的选项是向其指定一个年月，以显示当月的日历。

【用法示例】

（1）直接使用 cal 命令可以查看当前月份的日历：

```
#使用 cal 命令查看当前月份的日历
```

```
$ cal
   September 2010
Su Mo Tu We Th Fr  Sa
          1  2  3   4
 5  6  7  8  9 10  11
12 13 14 15 16 17  18
19 20 21 22 23 24  25
26 27 28 29 30
```

命令会以反白的形式显示当前日期在日历中所处的位置（下划线所示）。

（2）用户也可以向 cal 命令指定要查看的年月，例如要查看 2009 年 2 月的日历：

```
#查看指定月份的日历
$ cal 2 2009
   February 2009
Su Mo Tu We Th Fr  Sa
 1  2  3  4  5  6   7
 8  9 10 11 12 13  14
15 16 17 18 19 20  21
22 23 24 25 26 27  28
```

3.4.3　修改日期时间命令 date 和 clock

由于设备的误差，有时需要手动修改系统日期和时间，修改系统时间可以使用命令，也可以使用网络同步的方式修改（将在第 8 章介绍）。本小节将简单介绍如何使用命令修改系统时间。

小知识：Linux 系统使用了两个时钟：其一是由硬件设备控制的硬件时钟，另一个是由操作系统控制的系统时钟（这两个时钟都采用 24 小时制）。因此如果需要修改系统时间，就应该同时修改硬件时钟和系统时钟。

（1）修改时间之前可能需要先修改时区，在 RHEL5.3 中，修改时区可以使用其自带的 setup 等工具，也可以使用命令 tzselect，按提示选择所在的时区即可。

（2）利用 date 命令的 s 选项可以修改系统时钟中的日期和时间。例如要修改当前的系统时钟的日期为 2008 年 6 月 25 日：

```
#使用 s 选项修改当前系统的日期
# date -s 06/25/08
```

由于修改系统时间属于管理工作之一，因此应该以 root 身份执行以上命令。

（3）完成时间的修改之后，可以继续使用 date 命令的 s 选项修改时间。例如修改当前系统的时间为 15:30:00：

```
#使用 s 选项修改当前系统的时间
# date -s 15:30:00
Wed Jun 25 15:30:00 CST 2008
```

（4）由于使用 date 命令修改的日期和时间都只保存在系统时钟内，因此还需要将时间同步到硬件时钟。同步系统时钟到硬件时钟可以使用 clock 命令：

```
#将当前系统的时间写入硬件时钟
# clock -w
```

🔔**提示：** 由于操作系统关闭时，会自动使用系统时钟同步硬件时钟，因此如果不是常年开机的系统，也可以不同步硬件时钟。

3.5　联线用户命令

目前有许多 Linux 系统运行在多用户环境中（例如超级计算机、集群等环境），这些计算机的管理员可能还需要管理联线的用户。联线用户管理的内容包括查看联线的系统用户、与联线用户通信、断开联线的用户等，本节将简单介绍与联线用户相关的命令。

3.5.1　查看联线用户命令 who、finger 和 w

要管理联线用户，首先需要查看当前联线的用户。查看联线用户可以使用 who、finger 和 w 命令。虽然这 3 个命令都可以查看联线到系统的用户，但这三者之间有细微的差别。

（1）使用 who 命令查看当前系统中的用户：

```
#使用 who 命令查看联线到系统中的用户
# who
root    pts/1        2010-09-12 00:53 (192.168.100.21)
ljx     pts/2        2010-09-12 02:02 (192.168.18.237)
user1   pts/3        2010-09-12 00:53 (192.168.100.21)
wlh     pts/4        2010-09-12 02:02 (192.168.18.232)
```

who 命令输出了所有联线的用户、登录的终端时间和 IP 等信息。

（2）使用 finger 命令查看登录到系统的用户及相关信息：

```
#使用 finger 命令查看联线到系统的用户及详细个人信息
# finger
Login   Name    Tty    Idle  Login Time   Office     Office Phone
ljx             pts/2   5    Sep   12 02:02 (192.168.18.237)
root    root    *pts/1       Sep   12 00:53 (192.168.100.21)
user1   user1   pts/3   78d  Sep   12 00:53 (192.168.100.21)
wlh             pts/4   4    Sep   12 02:02 (192.168.18.232)
```

finger 输出了登录到系统的用户名称、终端和电话等详细个人信息，要查阅联线的用户及其个人信息，可以使用该命令。

（3）使用 w 命令查看正在使用系统的用户：

```
#使用 w 命令查看用户占用系统资源的情况
# w
 02:13:36 up 1 day,  1:41,  4 users,  load average: 0.00, 0.00, 0.00
USER     TTY    FROM            LOGIN@   IDLE    JCPU  PCPU WHAT
root    pts/1  192.168.100.21  00:53   0.00s   0.56s 0.01s w
ljx     pts/2  192.168.18.237  02:02   40.00s  0.04s 0.01s ping 61.**.*.*
user1   pts/3  192.168.100.21  00:53   808days 0.13s 0.13s -bash
wlh     pts/4  192.168.18.232  02:02   49.00s  0.03s 0.00s sleep 300
```

由以上输出结果可以看出，命令 w 功能很强大，不仅输出了用户及使用的终端，还输出了用户正在执行的命令及 CPU 占用时间等信息。

从以上 3 个命令的输出可以看出，who 命令主要用于一般性查看，finger 命令则侧重于用户的个人详细信息，而 w 命令则更侧重于输出用户使用系统的情况。管理员可以按需要使用这 3 个命令查看联线用户的不同信息。

3.5.2　与联线用户通信的命令 wall、write 和 mesg

查询到联线的用户及其详细信息之后，有时可能要向这些用户发送一些消息，这些消息可能是告知用户有关系统的一些情况，也可能是向用户发送其使用系统的情况等。按发送的消息不同，可以使用多个不同的命令，本小节将简单介绍与联线用户通信的相关命令。

（1）如果要向所有在线的用户发送公告、通知等消息，可以使用 wall 命令。例如要向使用系统的所有用户发送关机公告：

```
#使用 wall 命令向系统中的所有用户发送公告
# wall The system is going DOWN for system halt in 5 minutes!
```

此时登录到系统的用户都会得到以下提示：

```
Broadcast message from root (pts/1) (Sun Sep 12 02:40:14 2010):

The system is going DOWN for system halt in 5 minutes!
```

（2）要向登录到系统中的一个用户发送消息，可使用 write 命令。例如用户 ljx 要向 root 发送消息：

```
$ write root
#输入以上命令之后将会进入即时消息模式
Hello!
See it?
#此时按 Ctrl+C 键退出即时消息模式
```

输入命令和用户之后，将会进入命令的即时消息模式，此时用户可以继续输入消息，这些消息都会动态地显示到指定用户的终端屏幕。

在即时消息模式中，可以使用 Enter 键换行。当输入结束要退出时，可以使用组合键 Ctrl+C。

当用户输入以上消息时，root 用户终端屏幕将会显示如下内容：

```
#root 用户将显示其他用户发送的即时消息
#
Message from ljx@localhost.localdomain on pts/2 at 03:00 ...
Hello!
See it?
#EOF 为用户按下 Ctrl+C 键时得到的消息
EOF
```

（3）有时用户可能不希望被其他用户的消息打扰，此时可以使用命令 mesg n 拒绝其他用户发送来的消息。

如果对方拒绝消息，在发送时将会得到如下提示：

```
#当用户请求即时消息时被用户拒绝
$ write root
write: root has messages disabled
```

（4）如果用户使用了命令 mesg n 拒绝消息，可以使用 mesg y 重新接受消息。

3.5.3　断开联线用户命令 fuser

在一些特殊情况下，需要对系统进行紧急维护，这时应该将已经联线到系统的用户强制断开，并且不允许除 root 以外的用户登录。

（1）紧急维护时。首先需要强制断开用户连接的终端设备，这时可以使用 fuser 命令。例如要断开用户 user1 连接的终端设备：

```
#断开用户连接的终端设备
# fuser -k /dev/pts/3
#fuser 命令将通过结束终端的方式断开与用户的连接
/dev/pts/3:          5384
```

此时用户 user1 将会主动断开与系统的连接。

（2）紧急维护的另一种情况是禁止所有用户登录，这时可以在目录/etc 中新建一个名为 nologin 的空文件（逐一修改每个用户的 Shell 为/sbin/nologin，也可以禁止用户登录，但这种方法太繁琐），此时除 root 外的所有用户将无法登录到系统。

```
# touch /etc/nologin
```

此时系统将不会接受除 root 以外的用户登录，但是已经登录的用户将不会受到影响（即使将运行级别切换到单用户模式，系统也不会主动断开已经存在的用户连接），所以应该使用 fuser 将这些用户断开。

3.6　切换用户命令

在 Linux 系统中，root 用户拥有至高无尚上的权限，使用 root 用户登录可能会由于人为的误操作导致数据丢失（例如错误地删除了某个包含有重要数据的目录等）。因此一般情况下都不建议使用 root 用户登录系统，以免无意中损坏数据。通用的做法是使用普通用户登录并使用系统，当需要执行管理操作时，再切换到 root 用户执行管理操作。本节将简单介绍如何切换用户及获取 root 身份。

3.6.1　临时切换用户命令 su

su 命令用于临时切换到指定的用户。当用户使用此命令切换用户后，执行的操作将以切换后的用户身份执行，直到用户执行 exit 命令为止。

（1）例如要切换到 root 用户：

```
$ su root
```

```
Password:
#普通用户切换到 root 用户时，必须输入 root 用户的密码
#root 用户切换至普通用户时，可以不必输入密码
# whoami
root
```

上面这条命令演示了普通用户如何使用 su 命令切换到 root 用户，从第 2 条命令 whoami 的输出中可以看出，普通用户已经成功地切换到 root 用户。

注意：在 RHEL5.3 中，使用 su 命令虽然可以切换到 root 用户身份，但切换之后使用的环境变量仍然没有改变（有些系统也可以获得 root 用户的环境变量，例如 Fedora）。因此执行一些管理命令时，可能会提示没有找到命令，此时可以使用输入命令的全路径的方式执行相关命令（例如使用/sbin/ifconfig 替代 ifconfig 命令）。

（2）完成相关的功能后，要退出临时使用的用户，可以使用 exit 命令。例如退出临时使用的 root 用户：

```
#使用 su 命令切换用户后，可以使用 exit 命令退出
# exit
exit
#使用 whoami 验证是否成功退出
$ whoami
ljx
```

3.6.2　以 root 用户身份运行命令 sudo

su 命令虽然提供了使用 root 用户身份的方法，但是将 root 用户密码告诉许多人是很危险的，此时可以使用 sudo 命令。

sudo 命令工作时，先切换到 root 用户，并以 root 用户身份执行命令，然后返回到当前用户。当前用户使用 sudo 命令执行的操作取决于 root 用户在/etc/sudoers 文件中的授权，即任何管理命令都需要 root 授权，否则不能使用 sudo 执行。

例如要查看网络接口 eth0 的详细信息：

```
$ sudo /sbin/ifconfig eth0
#使用 sudo 命令时需要输入当前用户的密码（非 root 用户密码）
Password:
#验证成功后即可执行命令
eth0  Link encap:Ethernet  HWaddr 00:0C:29:B8:02:24
      inet addr:192.168.18.234  Bcast:192.168.18.255  Mask:255.255.255.0
      inet6 addr: fe80::20c:29ff:feb8:224/64 Scope:Link
……
```

sudo 命令不提供 root 用户的登录环境，因此上面的命令仍然使用全路径的方法运行 ifconfig 命令。

本小节仅简单介绍 sudo 命令的用法，关于 root 用户授权的相关内容请读者阅读相关安全说明和使用文档。

3.7　定位和查找

有时需要快速搜索文件、文档、命令等，Linux 为不同的内容提供了不同的快速搜索命令，使用这些快速搜索命令可以帮助使用者在最短时间内找到需要的内容。本节将简单介绍 Linux 系统中的快速搜索命令及其用法。

3.7.1　搜索命令 which

许多时候需要知道一个命令文件的位置（例如需要知道一个软件启动命令的具体位置，以便于确定软件安装位置），这时可以使用 which 命令。which 命令的功能是从当前的环境变量 PATH 保存的目录中，查找参数指定的命令位置并以绝对路径的方式输出。

例如需要查找命令文件 ifconfig 的位置：

```
#使用 which 查找命令文件 ifconfig 的位置
# which ifconfig
/sbin/ifconfig
```

查看环境变量 PATH 所包含的目录：

```
#显示环境变量 PATH 的值（即 which 的搜索目录）
# echo $PATH
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sb
in:/bin:/usr/sbin:/usr/bin:/root/bin
```

这是环境变量 PATH 的一个比较典型的值，当用户输入了一个命令之后，系统会在该变量所包含的目录中查找命令文件并执行找到的命令文件。

3.7.2　文件搜索命令 locate

locate 命令用于快速搜索文件和档案，其搜寻工作依赖于一个事先建立的索引数据库，这个索引数据库每天会自动更新一次，以便于搜集新增的文档。因此使用 locate 命令可能无法搜索到最近新增的文档。

说明： locate 命令来自于软件 slocate，不能正常使用此命令的系统可能需要手动安装此软件。

（1）例如要查找一个名为 iptables 的文件：

```
#查看文件名包含字符 iptables 的文件
# locate iptables
/etc/rc.d/init.d/iptables
/etc/rc.d/rc0.d/K92iptables
/etc/rc.d/rc1.d/K92iptables
……
```

（2）在使用 locate 查找文件时，也可以使用星号表示所有字符，点号表示所有单字符。例如要查找所有以字母 a 开头的文档：

```
#查看所有以 a 开头的文件
# locate a*
/root/access backup 20100711.log
/root/access backup 20100718.log
/root/access backup 20100725.log
/root/also supported by HP-UX
/root/anaconda-ks.cfg
```

（3）由于 locate 命令并非使用遍历所有目录的方法查找文档（即逐文件目录查找），因此可能查找到的结果并不是非常准确。如果要使搜索到的结果更加准确，可以先手动更新索引数据库，然后再查找文件。

手动更新索引数据库，可以使用以下命令：

```
#手动更新索引数据库
# updatedb
```

由于这个命令要搜集新增的文件，因此可能命令执行的时间会比较长。

3.7.3　特殊文件搜索命令 whereis

与前面几种查找命令相比，whereis 更加擅长于查找一些特殊的文件，这些特殊的文件包括二进制文件（通常是可执行文件）、说明文件和源代码等。同 locate 命令一样，whereis 命令也是通过查询索引数据库的方法来搜索文件的。

例如查找文件 iptables：

```
#从特殊文件中查找文件名中包含 iptables 的文件
# whereis iptables
iptables: /sbin/iptables /lib/iptables /usr/share/man/man8/iptables.8.gz
```

从上面的结果中可以看出，命令给出了 iptables 命令、库文件目录和 man 手册的详细位置。

3.7.4　关键字搜索命令 apropos

在使用 Linux 系统的过程中，许多时候可能想要完成某个特定的任务，而又不知道应该如何才能完成。这是许多初学者经常遇到的问题。在第 2 章中介绍了使用 man 命令的 k 选项在手册中查找命令的方法，除此之外，还可以使用命令 apropos 查找与任务相关的命令，apropos 命令与 man 命令的 k 选项类似，都是从 man 手册中查找关键字，并将找到的结果显示给用户。

例如要查找与进程相关的内容：

```
#在帮助文档中查找与 process 相关的内容
# apropos process
AF_LOCAL [unix]      (7)  - Sockets for local interprocess communication
```

```
AF UNIX [unix]       (7)  - Sockets for local interprocess communication
CPU CLR [sched setaffinity] (2)  - set and get a process's CPU affinity mask
CPU ISSET [sched setaffinity] (2)  - set and get a process's CPU affinity
mask
……
```

命令列出了查找到的函数和命令的用途及简要说明，用户可以通过此命令的输出结果查找需要的命令。

3.8　输入/输出相关命令

无论是在使用 Linux 过程中，还是在使用 Shell 编程过程，都不可避免地需要接触到与输入/输出相关的命令。Linux 系统提供了许多用于输入/输出的命令，利用这些命令可以完成更丰富的输入/输出功能。

3.8.1　回显命令 echo

echo 命令的功能是将参数指定的字符串输出到标准输出，由于功能特殊，通常形象地将其称为回显命令。回显命令是 Linux 系统中应用最广的输出命令之一，主要用来输出各类变量、提示信息等。

【命令格式】

```
echo [option] string
```

【常用选项】

- E：禁用转义字符（默认选项）。
- e：启用转义字符。
- n：输出结束后不换行。

这个命令相对比较简单，经常将这个命令和其他命令一起配合使用输出提示信息、错误信息等。

【用法示例】

（1）echo 命令最常用的是输出字符串，例如要将一个字符串输出：

```
#将指定的字符串输出（屏蔽转义字符）
$ echo "The first line.\nThe second line.\nField1\tField2"
The first line.\nThe second line.\nField1\tField2
```

（2）有时需要使用转义字符，让输出结果看起来更便于阅读，此时可以配合使用选项 e 启用转义字符。

例如使用转义字符“\n”换行，用“\t”分隔字段：

```
#使用选项 e 启用参数字符串中的转义字符
$ echo -e "The first line.\nThe second line.\nField1\tField2"
The first line.
The second line.
```

```
Field1  Field2
```

（3）有时需要获得用户的输入（例如某个操作需要用户批准），获得用户输入的提示信息通常都不会换行，这时可以配合使用选项 n。

例如输出提示信息且不换行：

```
#使用选项 n 输出提示信息并且不换行
$ echo -n "Please enter your name:"
Please enter your name:
```

echo 命令的 n 选项通常与读取输入命令 read 一起配合使用，read 命令将在下一小节中介绍。

（4）echo 还可以与特殊字符配合，在输出的字符串中添加变量、命令输出等内容，例如：

```
#在字符串中引用变量、命令输出
$ echo "Host $HOST_DOWN_NAME non-response,time is:`date +'%Y-%m-%d %T'`"
Host Web1 non-response,time is:2010-08-20 02:10:13
```

上面这个命令使用美元符号$引用变量，而使用反引号“``”引用命令的输出。

（5）echo 命令也可以与重定向配合输出错误信息（这个用法通常应用在 Shell 脚本中）。例如：

```
#将 echo 的输出以标准错误的方式输出
$ echo "Input error!"1>&2
```

如果使用错误重定向，可以发现上面这个命令的输出将变为标准错误。

3.8.2　接收用户输入命令 read

read 命令用于读取字符，读取的字符可以来自标准输入，也可以来自文件。这个命令广泛用于接收来自用户的输入，在可交互式脚本中经常用到。

【命令格式】

```
read [[option]parameter] name
```

【常用选项】

- a：将输入的文本按字段放入数组中（默认使用的分隔符为空格或制表符 Tab），数组的下标从 0 开始。
- d：指定结束符，当输入中出现指定的结束符时，read 命令将认为输入已经结束（默认使用换行为结束符）。
- e：从标准输入读取时，使用 Readline 库获取输入（此时用户可以使用退格键、方向箭进行简单的行编辑操作）。
- n：指定读取的字符数 n。
- p：将该选项指定的字符串输出到标准错误，并读取输入。
- r：将读取到的字符串按字段放到不同的变量中。
- s：先锁住屏幕，然后再从标准输入中读取字符。

- t：如果用户在指定的时间内没有输入，就结束读取状态。
- u：从指定的文件描述符中读取。

大多数情况下，read 命令都从标准输入读取用户输入，读取到数据后，read 命令会将数据放入指定的变量中。

当 read 命令读取的目标是一个文件时，如果没有特别指定，read 会使用换行符作为结束读取标记，即按行读取文件内容。

【用法示例】

（1）提示用户并接收输入

使用 read 命令接收来自用户的输入时，通常都使用 echo 命令提示用户。例如让用户输入自己的姓名：

```
#接收用户输入时，使用 echo 命令输出提示信息
$ echo -n "Please enter your name:";read NAME;echo "Hello $NAME"!
Please enter your name:Jhon
Hello Jhon!
```

上面的示例中，先使用 echo 显示提示信息，然后读取用户的输入并保存在变量 NAME 中，最后将结果输出。为了使输入光标与提示信息保持在同一行，应该使用选项 n 禁止 echo 输出提示信息后换行。

（2）读取多个数据并保存到数组中

有时来自用户的输入是一组并列关系的数据（例如人名、地名等），这些数据可能有特殊的用途，这时可以考虑将这些数据放入数组内。

例如要输入一组人名并将其放入数组内，然后输出：

```
#使用选项 a 将读取到的输入存入数组中，然后使用 echo 命令输出
$ echo -n "Please enter your friend's name:";read -a NAME;echo "Your
friends:${NAME[*]}"
Please enter your friend's name:Alix Alice Hope
Your friends:Alix Alice Hope
```

上面的示例中，read 命令先将用户的输入按空格分隔，并放入数组 NAME 内，最后将数组输出。

（3）指定输入结束符

默认情况下，read 命令使用换行符作为其接收输入的结束符，有时可能需要使用其他的字符作为输入结束符，可以使用选项 d 指定输入的结束符。

例如使用$作为输入结束符：

```
#使用选项 d 指定输入的结束符
$ echo -n "Please enter your name:";read -d "$" NAME;echo "Hello $NAME"!
Please enter your name:Jhon
Alix
$Hello Jhon
Alix!
#当输入中出现结束符时，read 命令将立即结束接收输入状态
```

（4）使用行编辑功能

当我们为脚本需要编写一个菜单时，必须要为用户输入提供行编辑功能，以免错误的

输入产生误操作。这时可以使用选项 e，允许用户在输入时使用简单的行编辑功能。

下面是一个简单的示例：

```
#使用选项 e 开启输入行编辑功能
$ echo -n "Please enter your message:";read -e MESS;echo "your message
is:$MESS"
Please enter your message:Good idea! I'll do.
your message is:Good idea! I'll do.
```

在这个示例中，用户输入时可以使用方向键、退格键修改已经输入的字符。

（5）指定接收的字符长度

有时可能需要用户输入一些格式化字符串（即具备一定格式的字符串，例如学号、手机号码等），此时可以配合使用选项 n 指定用户输入的字符长度。当用户输入的字符长度与指定长度相等时，read 将停止接收输入，并立即将接收的字符赋值给变量。

例如要求用户输入自己的手机号时，限定用户只能输入 11 位字符：

```
#使用选项 n 指定接收输入的字符串长度
$ echo -n "Please enter your phone number:";read -n 11 PHONE;echo -e "\n
Your phone number is:"$PHONE
Please enter your phone number:13321211212
#当用户的输入达到限定的长度时，read 命令将立即结束接收输入状态
Your phone number is:13321211212
```

（6）输出标准错误信息

编写交互式脚本时，可能需要输出错误信息并询问下一步要进行的操作，这时应该使用选项 p 将错误信息以标准错误的方式输出，以便于用户捕获错误信息并做出判断。

例如输出提示信息到标准错误，并接收用户的指示：

```
#使用选项 p 指定输出的标准错误
$ read -p "Unexpected error, press Y key to continue." Flag;echo "Being
restored, please wait..."
Unexpected error, press Y key to continue.y
Being restored, please wait...
```

（7）读取输入并保存到多个变量中

read 命令还提供了一个特殊的功能，将读取到的字符串按字段进行分隔，并保存到多个变量中（默认的字段分隔符为空格、制表符 Tab 和换行），这个用法与选项 a 类似。

例如要求用户输入 3 个人名并保存到不同的变量中：

```
#使用选项 r 将用户的输入保存到多个变量中
$ echo -n "Please enter the four string:";read -r V1 V2 V3 V4;echo -e
"first:$V1\nsecond:$V2\nthird:$V3\nfourth:$V4"
Please enter the four string:one two three four
first:one
second:two
third:three
fourth:four
```

上面这条命令接收用户输入的 4 个字符串，并分别存入 4 个变量然后输出。

如果字段分隔符不是空格，可以通过修改变量 IFS 的方法重新设置分隔符。例如设置

新的字段分隔符#，并接收用户的输入：

```
#通过修改变量 IFS 的方法指定新的字段分隔符
$ IFS="#";echo -n "Please enter the four variables:";read -r V1 V2 V3 V4;echo
-e "first:$V1\nsecond:$V2\nthird:$V3\nfourth:$V4"
Please enter the four variables:one#two#three#four
first:one
second:two
third:three
fourth:four
```

使用以上命令时，建议在修改 IFS 变量值之前，使用命令 Old_IFS=$IFS 保存默认分隔符，使用结束后使用命令 IFS=$Old_IFS 恢复默认的分隔符。

（8）锁住屏幕

有时需要让用户输入一些比较敏感的信息，把这些信息直接显示在屏幕上非常不安全，（例如用户需要输入密码等）。这时可以使用 read 命令提供的选项 s，先锁住当前用户的屏幕，待输入结束后再解锁屏幕。

例如要求用户输入密码：

```
#使用选项 s 在接收输入时锁住用户的屏幕
$ echo -n "Please enter your password:";read -s Pass;echo -e "\nYour password
is:"$Pass
#由于用户输入密码时，屏幕已经被锁住，因此无任何显示
Please enter your password:
Your password is:1234
```

（9）设置输入超时限制

有时需要为用户的输入设置时间限制（超时限制功能与引导选择菜单类似），即如果用户超过一定时间没有输入，则使用一个默认值替代用户的输入。使用 read 命令的 t 选项可以实现超时限制功能。

例如设置用户的输入时间为 8 秒：

```
#使用选项 t 设置用户输入的时间限制为 8 秒
$ SE=y;echo -n "Please select yes or no[y|n]:";read -t 8 SE;echo -e "\nInput
timeout,Use the default options[y]."
#此时不作任何输入，等待 8 秒
Please select yes or no[y|n]:
Input timeout,Use the default options[y].
```

在上面这个示例中，如果用户超过 8 秒没有输入 y 或 n，则使用默认值 y 替代用户的输入。

3.8.3　显示并保存文本命令 tee

tee 是一个比较常用的命令，它的作用是将获取到的数据（tee 命令获取到的数据通常来自管道）分为两个拷贝，一个拷贝输出到标准输出，另一个拷贝写入到指定的文件中。本小节将简单介绍 tee 命令的使用方法。

【命令格式】

```
…… | tee [options] filename
```

【常用选项】

- ❑ a：以追加的形式将数据写入到文件的结尾。
- ❑ i：写入数据时，不写入标准错误。

【用法示例】

（1）例如，将指定的网络接口卡信息输出到屏幕的同时也写入文件中：

```
#使用 tee 命令将来自管道的文本输出到标准输出，并写入文件 eth2_info 中
# ifconfig eth2 | tee eth2_info
eth2      Link encap:Ethernet  HWaddr 00:0C:29:EA:EF:47
         inet addr:192.168.0.2  Bcast:192.168.0.255  Mask:255.255.255.0
         inet6 addr: fe80::20c:29ff:feea:ef47/64 Scope:Link
……
#查看文件 eth2_info 的内容以验证 tee 命令的作用
# cat eth2_info
eth2      Link encap:Ethernet  HWaddr 00:0C:29:EA:EF:47
         inet addr:192.168.0.2  Bcast:192.168.0.255  Mask:255.255.255.0
         inet6 addr: fe80::20c:29ff:feea:ef47/64 Scope:Link
……
```

（2）有时并不希望 tee 命令使用覆盖的方式将数据写入到文件，这时可以使用选项 a，将数据以追加的形式写入文本结尾。例如：

```
#使用选项 a 将当前时间追加到文件结尾
# date +'%Y%m%d %T' | tee -a date_log
20100821 01:06:49
$ cat date_log
20100821 01:06:44
20100821 01:06:49
```

（3）tee 命令还提供了简单的筛选功能，有时得到的数据中不仅含有标准输出，还含有标准错误，此时可以使用选项 i 忽略标准错误。例如：

```
#使用 tee 命令写入时，忽略内容中的标准错误
# ls -l ab | tee -ia a.txt
```

3.8.4　邮件命令 mail

mail 是 Linux 系统中自带的电子邮件命令，通常可以用这个命令发送、阅读电子邮件。在使用脚本自动管理系统的过程中，管理员经常使用 mail 命令将脚本执行的结果或脚本日志发送到指定邮箱，以便于了解脚本的执行情况。本小节将简单介绍如何使用 mail 命令发送、阅读电子邮件。

1．发送电子邮件

【命令格式】

```
$ mail [option] [-s subject] [-c c_adress] [-b b_adress] user [-a file]....
```

【常用选项】

- n：不读入设置文件/etc/mail.rc（这个文件允许用户使用外部地址而不是系统自带的 sendmail 发送邮件）。
- s：设置邮件的主题信息。
- c：使用一个密件抄送列表。
- b：使用一个抄送列表。

【用法示例】

（1）发送邮件时，通常将邮件的正文放入一个文件内，然后使用重定向的方式读取并发送文件中的内容。例如要将文件 net_info 中的内容发送给用户 root 和 user1，并将邮件的主题设置为 net info：

```
#使用 mail 命令向用户 root 和 user1 发送主题为 net info 的邮件
#使用重定向的方法指定邮件的内容为文件 net info 的内容
$ mail -s "net info" root,user1 <net_info
```

在上面这个示例中，由于收件人的地址为系统中的用户，因此邮件将直接发送到用户的系统邮箱中。

提示：如果接收邮件的用户或地址有多个，可以使用逗号“,”和空格作为邮件列表的分隔符。

（2）使用 mail 发送邮件时，除了使用系统中的用户作为地址外，还可以使用外部邮箱作为地址（前提条件是网络处于可用状态）。例如：

```
#使用一个外部邮箱作为地址
$ mail -s "net info" root,user1,ne****@163.com <net_info
```

这样 ne****@163.com 就可以收到一个来自 user@localhost.localdomain 的邮件（由于 user@localhost.localdomain 并不是一个可以返回的邮箱地址，因此某些邮箱可能会拒收此类邮件）。

2. 查看邮件

有时系统中的用户可能会收到如下提示信息：

```
You have new mail in /var/spool/mail/root
```

这个提示信息将告知用户接收到新的系统邮件。

（1）可以使用命令 mail 查看邮件列表：

```
#使用 mail 命令查看当前用户的邮件列表
# mail
#以下为交互式页面内容
Mail version 8.1 6/6/93.  Type ? for help.
"/var/spool/mail/root": 8 messages 8 unread
>U  1 root@localhost.local  Sun Aug 22 07:33  36/2571  "netstat"
 U  2 logwatch@localhost.l  Sun Aug 22 07:42  44/1623  "Logwatch for
localhos"
 U  3 root@localhost.local  Sun Aug 22 07:56  36/2588  "netstat"
```

```
 U  4 root@localhost.local  Sun Aug 22 08:36  36/2556  "test"
 U  5 root@localhost.local  Sun Aug 22 08:37  36/2556  "test"
 U  6 root@localhost.local  Sun Aug 22 08:38  36/2557  "test1"
 U  7 root@localhost.local  Sun Aug 22 08:42  36/2557  "test1"
 U  8 root@localhost.local  Sun Aug 22 08:47  36/2557  "test1"
&
```

在上面这个示例中，前两行先打印出了 Mail 的版本号、如何获取帮助信息、当前邮箱中邮件数目等内容。后面的邮件列表中，符号“>”指示当前光标的位置， U 表示未读邮件，其后分别是邮件序号、发件人、发件时间、邮件主题等内容。最后一行的“&”是邮件命令提示符，用户可以在此提示符后面输入命令并查看邮件。

（2）在邮件命令提示符中常用的命令如下。

- h：查看邮件的标题。
- d：使用该命令加序号可以删除指定的邮件。
- f：查看当前光标指向的邮件。
- n：跳转到指定序号的邮件。
- q：退出并将已经阅读的邮件存入文件~/mbox 中。
- x：退出。
- !：允许使用 Shell 命令。

用户使用以上命令打开邮件后，可以使用 Enter 键或空格键翻页，阅读完邮件后将会自动返回邮件列表。

（3）通常建议退出时使用命令 q，以便于将已阅读的邮件存入文件~/mbox 中。如果要重新阅读 mbox 中的邮件，可以使用选项 f。例如：

```
#使用 f 选项查看文件~/mbox 中的已读邮件
# mail -f
Mail version 8.1 6/6/93.  Type ? for help.
"/root/mbox": 81 messages
>  1 root@localhost.local  Sun Aug 22 07:33  36/2572  "netstat"
   2 logwatch@localhost.l   Sun  Aug  22  07:42   44/1624   "Logwatch  for
localhos"
   3 root@localhost.local  Sun Aug 22 07:56  36/2589  "netstat"
   4 root@localhost.local  Sun Aug 22 08:36  36/2557  "test"
……
&
```

（4）管理员也可以使用选项 f 查看其他用户的邮件，例如：

```
#使用 f 选项查看用户 user1 的未读邮件
# mail -f /var/spool/mail/user1
Mail version 8.1 6/6/93.  Type ? for help.
"/var/spool/mail/user1": 5 messages 1 new 5 unread
 U  1 root@localhost.local  Sun Aug 22 07:33  36/2571  "netstat"
 U  2 root@localhost.local  Sun Aug 22 07:36  37/2608  "netstat"
 U  3 root@localhost.local  Sun Aug 22 07:56  36/2588  "netstat"
 U  4 root@localhost.local  Sun Aug 22 09:18  14/640
>N  5 root@localhost.local  Sun Aug 22 09:19  18/646
&
```

3.8.5　启动新 Shell 命令 exec

exec 命令用于启动一个新的 Shell，并在新 Shell 中执行指定的命令。在实际应用中，exec 命令常被用来处理文件描述符，此时（也只在此时）exec 不会启动新 Shell 替代当前 Shell。

【命令格式】

```
exec shell-command
```

exec 执行时，会使用新的 Shell 替代当前 Shell（而不是启动子 Shell），如果没有特别指明，exec 将在命令执行完成后退出启动的新 Shell。

【用法示例】

（1）下面是一个使用文件描述符读取文件的前 3 行的示例：

```
#使用 exec 操作文件描述符的例子
#使用保持指针的方法读取文件的前 3 行
$ exec 3<ping.txt;read -u 3 LINE;read -u 3 LINE1;read -u 3 LINE2;echo -e
"$LINE \n $LINE1 \n $LINE2";exec 3<&-
PING 61.139.2.69 (61.139.2.**) 56(84) bytes of data.
 64 bytes from 61.139.2.**: icmp_seq=1 ttl=128 time=5.46 ms
 64 bytes from 61.139.2.**: icmp_seq=2 ttl=128 time=5.22 ms
```

上面这个示例中，先将文件描述符 3 指向文件 ping.txt（此处使用的是“3<”表示一个输入文件描述符 3）。然后使用 read 命令的选项 u，从文件描述符 3 中读取 3 次，并使用命令 echo 输出读取到的内容。最后的命令 exec 3<&-表示关闭文件描述符 3。

由于在读取文件内容的过程中，并没有关闭文件描述符，因此 read 命令读取到的内容是文件的前 3 行。为了更深入理解文件描述符的作用，读者可以不使用文件描述符对文件执行读取并输出，查看其结果有何不同。

（2）下面是一个输出文件描述符的例子：

```
#使用 exec 命令操作输出文件描述符
#使用保持（写入）指针的方法实现文本追加功能
$ exec 4>name list;echo -n "Please enter a string:";read NAME1;echo -n "Enter
the second string:";read NAME2;echo -e "The first string:"$NAME1"\nThe
second string:"$NAME2 >& 4 ;exec 4>&-;cat name_list
Please enter a string:Jhon
Enter the second string:Alix
#使用 cat 命令查看文件 name_list 的内容
$ cat name_list
The first string:Jhon
The second string:Alix
```

注意：在关闭文件描述符时，必须指明文件描述符，如果将文件描述符 0、1、2 关闭则可能退出当前 Shell。

3.9 小　　结

- ❑ 3.1 节主要讲述了 Linux 系统中几个使用频率非常高的基本命令，这些命令在使用系统的过程中经常用到，因此应该特别注意。
- ❑ 3.2 节简单介绍了 Linux 系统中几个常用的文件操作命令及其常见的用法。
- ❑ 3.3 节介绍了几个常见的与文本内容相关的命令，其中查看文件内容的相关命令使用频率较高，读者应该掌握这些命令的使用方法。
- ❑ 3.4 节简单讲解了 Linux 系统中与日期时间相关的命令。虽然日期时间命令使用较少，但读者应该了解这些命令。
- ❑ 3.5 节讲述了联线用户相关的命令，包括查看联线用户、与联线用户通信、断开联线用户等内容。
- ❑ 3.6 节讲述了切换用户相关的命令，包括临时切换用户命令 su，以及以 root 用户身份执行命令等内容。合理地使用这些命令，有助于增强系统的安全性。
- ❑ 3.7 节简单介绍了 Linux 系统中用于查找和定位的相关命令。初学者应该了解这些命令的用途，以便于在需要时使用。

对于大多数人而言，本章讲解的许多命令及命令的常见用法都需要掌握，因此建议读者多练习这些命令，以便在需要时能够熟练运用。

第 4 章　Linux 命令中的特殊字符和正则表达式

许多时候需要筛选有用信息，这些信息可能包含在文件名称、命令输出、文本文件中。如果只记得筛选内容中的一部分或不连续的几部分，这时要查找或引用信息会非常困难，为了解决这个问题，Linux 系统引入一些特殊字符和正则表达式。

本章主要涉及的知识点如下。

- 使用符号引用字符串、变量、函数、命令输出。
- 使用符号“*”、“?”和“[]”匹配文件名称。
- 在多条命令中使用逻辑运算符和括号改变命令执行顺序。
- 使用正则表达式对文本进行匹配查找。

提示：在 Windows 中通常将一些正则表达式的基本字符称为通配符，而在 Linux 操作系统中人们则将这些基本字符称为元字符。使用多个元字符组成的查找表达式称为匹配模式，使用匹配模式查找的过程则称为（模式）匹配。

4.1　命令中的特殊字符

在 Linux 系统中有很多字符具有特殊意义，这些字符通常被用来引用、匹配（作用与通配符相同）、改变命令运行顺序等。本节中将使用实例讲解这些特殊字符及其用法。

4.1.1　字符串引用符双引号和单引号

在使用 Linux 的过程中，经常需要引用一些文本、变量、函数等，为了区分这些引用，需要使用引用符号。常见的引用符号有双引号、单引号、倒引号（有时也称反引号）和美元符号等，其中双引号和单引号的主要作用是引用字符串，避免引起误解。本小节将通过实例讲解双引号和单引号的使用方法。

1．双引号

双引号“""”通常用来表示引用一个字符串，有时能屏蔽一些标点等特殊字符。在命令中使用双引号时，一般表示引用的部分是一个不可分割的整体。

通常可以理解为所有放入双引号中的内容都按照字符串来处理，除非有另一个引用号告知某个特殊字符串具有特殊含义（例如变量等）。

（1）屏蔽特殊字符

如果要在命令中使用一个比较长的字符串作为参数，通常应该将这些字符串放入双引

号内，避免 Shell 对字符串中的某些字符产生误解。

例如要输出多个人名组成的字符串：

```
#输出由空格分隔的一组人名
$ echo Jhon Alix Albert Jack
Jhon Alix Albert Jack
```

上面这个人名组成的字符串输出没有问题，但是下面这个字符串却有问题：

```
#使用 echo 命令输出的字符中存在特殊字符分号时，引起了误解
$ echo Jhon Alix Albert Jack;Mary Roman
Jhon Alix Albert Jack
-bash: Mary: command not found
```

上面这个示例中包含了特殊字符分号“;”（分号表示一个命令的结束、另一个命令的开始），因此引起了系统误解。像这种情况一般应该使用双引号表示字符串的引用：

```
#使用双引号屏蔽字符串中的特殊字符
$ echo "Jhon Alix Albert Jack;Mary Roman"
Jhon Alix Albert Jack;Mary Roman
```

这时正确地输出了整个字符串。

（2）在双引号中使用其他引用符

如果要在双引号引用的字符串中引用，应该使用其他引用符号。例如使用“$”引用变量：

```
#先定义变量，然后再使用变量引用符将变量输出
$ NAME="Mary"
$ echo "Jhon Alix Albert Jack;Mary Roman $NAME"
Jhon Alix Albert Jack;Mary Roman Mary
```

除了变量引用符之外，还可以使用命令输出引用符：

```
#使用反引号引用命令输出
$ echo "Now time:`date`"
Now time:Thu Jul  7 11:23:43 CST 2011
```

（3）在命令中使用双引号

在一些命令中还经常使用双引号避免 Shell 误解选项的含义等。例如：

```
#使用 grep 命令查找变量 NAME 中是否包含 Jhon Alix
$ NAME="Jhon Alix Albert Jack;Mary Roman Mary"
#由于没有使用双引号，grep 将参数 Alix 当成文件处理，从而引起误解
$ echo $NAME | grep Jhon Alix
grep: Alix: No such file or directory
-bash: echo: write error: Broken pipe
$ echo $NAME | grep "Jhon Alix"
Jhon Alix Albert Jack;Mary Roman Mary
```

上面的例子中，命令 grep 将 Alix 当作一个文件来处理，从而引起了误会。

像上面这样的例子，在表示引用一个字符串时，应该使用双引号。

2．单引号

单引号“' '”的使用方法和作用与双引号一致，都是将引号中的内容都当作字符串来

处理。不同的是单引号通常在一些特殊命令中与双引号配合，表示引用中的引用（Shell 中不允许一条命令中的两个引号都使用双引号或单引号）。

（1）使用上面的示例：

```
#在 grep 命令中使用单引号，表示参数是一个整体
$ echo $NAME | grep 'Jhon Alix'
Jhon Alix Albert Jack;Mary Roman Mary
```

（2）读者可能会觉得没有必要引入单引号，然而在一些很特殊的例子中，将两个双引号嵌套使用会被 Bash 认为是不合法的，此时应该考虑配合使用单引号。例如：

```
#使用 awk 命令调用 system 执行 Shell 命令
$ awk 'BEGIN{system("date +'%Y%m%d%t%T'")}'
20100824        08:29:17
```

上面这个示例中，函数 system 中的命令应该放入双引号内，命令 date 的格式化字符串原本也应该放入双引号内，由于不能两个双引号嵌套使用，因此此处采用的是单引号。

4.1.2　命令引用符反引号

反引号“``”有时也称为倒引号，这个键通常位于键盘的 Tab 键上方、数字键 1 的左边。反引号主要用来引用一些函数、命令输出等，执行时系统会尝试将反引号中的内容当作一个命令去执行。

提示：如果在一条命令中出现了反引号，系统会首先执行反引号内的命令。

（1）在本章之前有一个使用反引号的例子：

```
#将 cat 命令的执行结果作为 ls 命令的参数
$ ls -l `cat /etc/shells `
-rwxr-xr-x 1 root root  735004 2009-01-22 /bin/bash
lrwxrwxrwx 1 root root       4 08-15 10:57 /bin/csh -> tcsh
……
```

将命令 cat /etc/shells 放入反引号中，表示这是一个命令而不是一个参数。

（2）在字符串中嵌套和使用一个命令的情况经常用到：

```
#使用倒引号引用当前系统的时间
$ echo "Now time:`date +'%T'`"
Now time:08:50:55
```

（3）可以在反引号中引用更长的命令：

```
#使用 who 和 awk 命令将联线用户输出给 echo 命令
# echo "Are using this system users:`who | awk '{ARR[FNR]=$1;}END{printf
"%s %s\n",ARR[1],ARR[2];}'`"
Are using this system users:root user1
```

4.1.3　变量引用和命令转换符美元符号

美元符号“$”表示引用一个变量，引用一个变量时只需要在变量名称前面加上“$”符号即可。

（1）下面是一个使用变量的例子：

```
#使用一个变量保存当前的日期并输出
$ date=`date`
$ echo "Host time is:$date"
Host time is:Mon Aug 23 10:14:59 PDT 2010
```

（2）美元符号“$”还可以用来置换一个命令，使用美元符号置换命令时，Bash 会将相应的命令放入子 Shell 中执行。

使用上面的示例：

```
#使用命令转换符引用命令输出
$ ls -l $(cat /etc/shells)
-rwxr-xr-x 1 root root  735004 Oct 21  2008 /bin/bash
lrwxrwxrwx 1 root root       4 Jul 26 04:40 /bin/csh -> tcsh
……
```

提示：在不同的系统中，命令转换符的使用方法可能会有所差别，有的系统可能需要在转换符和置换命令之间加入一个空格才能执行。

4.1.4　反斜线屏蔽符

许多时候需要在输出中使用这些特殊字符本身，这时应该使用反斜线“\”（也称反斜杠）屏蔽其特殊的含义。

（1）例如要使用美元符号表示货币：

```
#使用反斜线屏蔽输出字符串中的$
$ echo "Please pay \$15.50 ."
Please pay $15.50 .
```

（2）当输出内容中含有引号时，应该使用反斜线屏蔽引号的特殊含义：

```
#使用反斜线屏蔽输出字符串中的引号
$ echo "Tom:\"What is your name?\""
Tom:"What is your name?"
```

4.2　文件名通配符

在使用 Linux 系统的过程中，许多时候都需要查找一个文件（例如查找服务的配置文件等），如果只能记住文件名的一部分，查找时就显得非常困难。这时可以考虑使用文件名通配符，本节将通过实例详细讲解文件名通配符的使用方法。

4.2.1　单字符匹配元字符“?”

元字符“?”表示匹配文件名中任意一个字符，连续使用多个元字符“?”可以表示多个任意字符。

（1）例如要在当前目录下查找有两个字符且第 1 个字母是 a 的文件：

```
#使用 a?表示以 a 开头且有两个字符的文件名
$ ls -l a?
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 aa
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 ab
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 ac
……
```

（2）也可以使用多个“?”，例如要查看当前目录下由字母 a 和任意两个字符组成的文件名：

```
#查看当前目录下由字母 a 和任意两个字符组成的文件名
$ ls -l a??
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abc
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abd
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abe
……
```

（3）要查看当前目录下所有第 3 个字符是 c 的文件：

```
#查看当前目录中所有第 3 个字符是 c 的文件
$ ls -l ??c?
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abcd
-rw-r--r-- 1 user1 user1 347 Aug 24 08:25 abce
-rw-r--r-- 1 user1 user1 347 Aug 24 08:25 abcf
……
```

4.2.2　多字符匹配元字符“*”

元字符“*”表示匹配文件名中的任意字符串。匹配的字符串长度可以是零到一个字符，也可以是长度很长的多个字符组成的字符串。

（1）例如要查看当前目录下以字母 c 结尾的所有文件：

```
#查看当前目录下以字母 c 结尾的所有文件列表
$ ls -l *c
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abbc
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abc
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 ac
……
```

（2）例如要查看当前目录下所有以字母 ee 开头并以字母 c 结尾的文件：

```
#查看当前目录下以 ee 开头，并以 c 结尾的文件列表
$ ls -l ee*c
```

```
-rw-r--r-- 1 user1 user1 347 Aug 24 08:58 eec
-rw-r--r-- 1 user1 user1 347 Aug 24 08:58 eedac
-rw-r--r-- 1 user1 user1 347 Aug 24 08:58 eedbac
……
```

从上面的例子中可以看出，元字符“*”既可以匹配零个字符，也可以匹配多个字符。

（3）元字符“*”也可以与其他的元字符配合一起使用。例如要在当前目录中查看文件名倒数第 2 个字符为字母 c 的所有文件：

```
#查看当前目录中倒数第 2 个字符是 c 的所有文件
$ ls -l *c?
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abcd
-rw-r--r-- 1 user1 user1 347 Aug 24 08:25 abce
-rw-r--r-- 1 user1 user1 347 Aug 24 08:25 abcf
……
```

（4）在当前目录中查看第 3 个字符为字母 c 组成的所有文件：

```
#查看当前目录中第 3 个字符为 c 的所有文件
$ ls -l ??c*
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abc
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abcd
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abcde
……
```

4.2.3　字符范围匹配符“[]”

范围匹配符号“[]”通常用于匹配一个字符范围，其表现形式可以是减号“-”表示的字母和数字的范围，也可以是几个字符的组合。

（1）如果范围匹配符中出现的是几个字符的组合，表示匹配其中的任意一个字符。例如要在当前目录中查看以字母 klsyz 中的任意一个开头的所有文件：

```
#查看当前目录中以 klsyz 中任意字母开头的所有文件
$ ls -l [klsyz]*
-rw-r--r-- 1 user1 user1 347 Aug 24 09:16 kdwed
-rw-r--r-- 1 user1 user1 347 Aug 24 09:22 klsdsf
-rw-r--r-- 1 user1 user1 347 Aug 24 09:16 ldsfs
……
```

（2）使用多个通配符可以进行更加复杂的匹配查找。例如要在当前目录中查看以字母 a 开头且倒数第 2 个字符为字母 cde 中任意一个的所有文件：

```
#查看以字母 a 开头且倒数第 2 个字符为 cde 中任意一个字母的所有文件
$ ls -l a*[cde]?
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abcd
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abcde
-rw-r--r-- 1 user1 user1 347 Aug 24 08:21 abcdef
……
```

（3）使用符号“-”可以匹配一个字母或数字范围。例如要在当前目录下查看文件名中

含有数字的所有文件：

```
#查看当前目录中文件名中包含数字的所有文件
$ ls -l *[0-9]*
-rw-r--r-- 1 user1 user1 347 Aug 24 09:35 0e
-rw-r--r-- 1 user1 user1 347 Aug 24 09:35 0e7
-rw-r--r-- 1 user1 user1 347 Aug 24 09:34 db9sdf
……
```

上面的命令中[0-9]表示匹配 0 到 9 之间的任意数字。

（4）要查看当前目录下文件名由两个字符组成且第 2 个字符是字母的所有文件：

```
#查看文件名由两个字符组成且第 2 个字符为字母的所有文件
#[a-Z]表示所有小写字母、大写字母
$ ls -l ?[a-Z]
-rw-r--r-- 1 user1 user1 347 Aug 25 04:43 0E
-rw-r--r-- 1 user1 user1 347 Aug 25 04:43 6R
-rw-r--r-- 1 user1 user1 347 Aug 25 04:43 ad
……
```

上面的命令中，[a-Z]表示所有字母，也可以使用[a-zA-Z]表示所有字母。

4.2.4　排除范围匹配符“[!]”

排除范围匹配符“[!]”表示不匹配符号内出现的字符组合或字母数字范围。使用时感叹号“!”只能放置在要排除的字符串首。

（1）例如要查看当前目录下文件名只有两个字符并以数字开头且第 2 个字符不是数字的所有文件：

```
#查看文件名有两个字符且第 1 个字符为数字、第 2 个为非数字的所有文件
$ ls -l [0-9][!0-9]
-rw-r--r-- 1 user1 user1 347 Aug 24 09:35 0e
-rw-r--r-- 1 user1 user1 347 Aug 24 09:54 1j
-rw-r--r-- 1 user1 user1 347 Aug 24 09:54 8d
……
```

（2）要查看当前目录下文件名只有两个字符且两个字符都不是字母的所有文件：

```
#查看文件名有两个字符且文件名不包含字母的所有文件
$ ls -l [!a-Z][!a-Z]
-rw-r--r-- 1 user1 user1 347 Aug 24 10:02 ^^
-rw-r--r-- 1 user1 user1 347 Aug 24 09:39 35
-rw-r--r-- 1 user1 user1 347 Aug 24 10:02 %4
……
```

在上面的命令中[!a-Z]表示排除小写字母 a～z 和大写字母 A～Z，即排除所有字母。

注意：由于文件名中可能还存在其他字符（这些文件可能并不在查找范围之内），因此在使用范围匹配和排除范围匹配时，应该优先考虑使用排除范围匹配。

4.3　多条命令中的逻辑运算符和括号

使用多条命令执行任务时，两条相邻的命令之间可能会存在某种依赖关系，例如后一条命令是否需要执行，依赖于前一条命令是否能够成功执行。一个典型的应用实例是检查主机间的连通性，并根据检查结果执行相应的命令（例如无法连通时发送警告消息等）。此时两个命令间的关系类似于 C 语言中的 if 语句，不同的是 if 语句使用的是条件判断，而此处为前一个命令的执行情况。

Linux 系统提供了两个逻辑运算符：逻辑或“||”和逻辑与“&&”，还提供了两个括号：小括号“()”（也称圆括号）和大括号（也称花括号）“{}”用于控制多个命令执行顺序。本节将通过实例讲解如何使用逻辑运算符和括号控制命令执行顺序。

4.3.1　逻辑或“||”

【运算符格式】

```
command1 || command2
```

执行命令时，先执行命令 command1，如果 command1 执行成功，则跳过命令 command2 执行后面的内容，执行失败就会执行命令 command2。

逻辑或会尽量让两端的命令有一个执行成功，这与逻辑或的含义相同。

【用法示例】

（1）下面是一个读取邮件的例子：

```
#如果执行 mail 命令失败，则使用 echo 命令输出错误信息
$ mail -f /root/mbox || echo "Permission denied"
/root/mbox: Permission denied
Permission denied
```

使用 mail 命令读取 root 的历史邮件，如果失败，则显示提示信息。然而上面的示例中提示信息有两条，可以使用如下方法屏蔽错误提示：

```
#将 mail 命令的错误信息输出到系统池
$ mail -f /root/mbox 2>/dev/null || echo "Permission denied"
Permission denied
```

（2）逻辑或在管理和维护系统的过程中有时很有用。例如测试两个主机间的连通性，如果失败则返回提示：

```
#使用 ping 命令发送 3 个数据包测试本机与 125.1**.***.*的连通性
#如果连通性测试失败，则输出相关提示信息
$ ping 125.1**.***.* -c3 &>/dev/null || echo "Host Web1 network connection
is lost."
Host Web1 network connection is lost.
```

上面的命令中先使用 ping 命令检查与主机 Web1 之间的连通性，如果连通性测试失败，则输出提示信息。

🔔提示：逻辑操作符的运行原理与命令的退出状态有关，关于命令退出状态将在第 17 章中介绍。

4.3.2　逻辑与“&&”

【运算符格式】

```
command1 && command2
```

系统运行上面的命令时，会先执行命令 command1，如果执行成功，则会继续执行命令 command2；如果执行失败，就会跳过 command2。

逻辑与会尽量让两端的命令都执行成功，如果第 1 个命令执行失败，无论第 2 个命令结果如何，整个表达式都为假，因此将会忽略命令 command2。

【用法示例】

（1）例如为移动文件添加提示信息：

```
#使用 mv 命令移动文件，如果成功，则输出提示信息
$ mv ~/rm.txt ~/test/ && echo "Successfully moving files."
Successfully moving files.
```

（2）在管理和维护系统时，逻辑与可以用于当测试正常时向用户输出提示信息。例如检查主机 Web1 服务器是否能正常提供服务并输出提示信息：

```
#使用 wget 命令测试 Web1 服务器是否能正常服务
#如果 Web1 服务器正常，则输出提示信息
$ wget http://125.1**.***.* &>/dev/null && echo "`date +'%Y%m%d  %T%t'`  Host
Web1 service running."
20100824  12:16:25       Host Web1 service running.
```

上面的命令中先使用命令 wget 检查 Web1 能否正常提供服务，如果正常，则输出当前时间和提示信息。

4.3.3　括号

Linux 系统提供了两个括号：圆括号“()”和花括号“{}”，通常称为小括号和大括号。虽然这两个括号使用方法相同，功能却不一样。本小节将简单介绍这两个括号的使用方法。

【括号的基本格式】

```
(command1;command2;command3...)
{command1;command2;command3...}
```

Linux 系统遇到括号时，会按顺序先执行括号内的命令，直到括号内的所有命令执行结束，然后再返回执行其他命令。

花括号的使用方法与圆括号基本一致，不同的是花括号内的所有命令都会被放入子 Shell 中执行。通常情况下使用花括号将一组完成特定功能的命令放在一起执行（其实质是函数），花括号的使用方法将在第 17 章中详细介绍。

提示：通常将括号与命令逻辑操作符配合使用，以达到执行多条命令的目的。

【用法示例】

下面是一个使用圆括号监控系统的例子：

```
#使用 ping 命令判断与主机 125.1**.***.*的连通性
#如果连通性测试失败，则将信息输出的同时发送给 root 用户
$ ping 125.1**.***.* -c3 &>/dev/null || (echo "Host Web1 network connection
is lost." | tee network_info ;mail -s "network error" root <network_info ;rm
-rf network_info)
Host Web1 network connection is lost.
```

在上面这个示例中，将括号与逻辑操作符配合使用，从而达到了命令执行失败后执行多个命令的目的。

4.4　命令中的正则表达式

4.2 节中介绍了文件名中的通配符，使用文件名通配符模糊查找文件非常高效。而如果要在命令输出或文本中筛选内容时使用模糊查找，就需要使用正则表达式。正则表达式是一套由多个元字符组成的模糊查找模式，使用正则表达式可以快速查找和定位文本中指定的内容。

Linux 系统中的许多工具都可以使用正则表达式，这些工具包括 grep、awk、sed 和 Vi 等。本节将简单介绍正则表达式的元字符及其常见的正则表达式组合，关于正则表达式的使用示例将在后续章节中介绍。

提示：本节中讲述的元字符和匹配模式在一些系统和命令中用法可能会有所不同，请阅读相关文档以获得帮助或使用其他功能相近的模式替代。

4.4.1　单字符匹配符“.”

正则表达式主要由一些元字符和匹配模式组成，使用这些功能强大的元字符和匹配模式可以快速查找和定位字符串、单词和特定的模式等。从本小节起将一一介绍常用的元字符和匹配模式。

单字符匹配符“.”可以匹配任意单个字符，这个字符的功能与文件名匹配符中的“?”功能相同。

要使用正则表达式查找文本，首先需要使用元字符组成一个查找模式。与文件名通配符一样，查找模式由精确匹配字符和正则表达式的元字符组成。

（1）使用查找模式时，通常将其放入两个斜杠“//”中，然后再放入命令中。例如要在一个文本中查找匹配模式“/.i...../”，可能出现的匹配选项有：

```
#匹配模式可以匹配任意位置的文本
Jiangsu…
Sichuan…
Liulu…
Lixia…
…li…
……
```

能与此匹配的文本内容还有很多，只要小写字母没有出现在行首，或行尾的最后有 5 个字符都能匹配，即使要匹配位置的字符是空格也能完成匹配。因此在单独使用单字符匹配符时，应该尽量扩展已知字符，比如使用一个已知的格式化字符串，例如学号、产品编号和身份证号等，以保证查找的准确率。

（2）例如对一个格式化字符串学号使用单字符匹配符“/27210103../”，可能的匹配结果如下：

```
2721010301…
2721010305…
2721010307…
……
```

4.4.2　单字符或字符串重复匹配符“*”

单字符或字符串重复匹配符“*”用来匹配单个字符或一个字符串序列的一次或多次重复出现。

这个匹配符很灵活，例如使用“/bo*/”匹配时，可能匹配到的结果有：

```
boooooooooooo…
bookokokokokokok…
bokkkkkkkkkkkkk…
…
```

可见这是一个很灵活的匹配，在匹配时可以是已经出现的字符或字符串，也可以是没有出现过的字符或字符串重复。

4.4.3　行首匹配符“^”

行首匹配符“^”用于在匹配中指示行首位置字符串或模式，因此使用时需要将行首匹配符放在要匹配的字符串或模式的前面。

（1）例如对命令 ls -l 使用行首匹配“/^l/”，可以匹配的结果有：

```
lrwxrwxrwx …
lrwxrwx--x …
lrwx--x--x …
……
```

提示：由于正则表达式不具备匹配文件名的功能，因此不能对 ls -l 命令直接使用以上匹配模式。可以将上面的模式交由一些筛选命令处理，例如交由 grep 命令时可以写作：ls -l | grep /^l/。关于这些模式的详细使用示例将在后续章节中介绍。

（2）在一些规范化的文本中，行首往往保存的是关键信息，类似数据库里的主键（记录的唯一标志字段，例如学号、产品编号等）。对这类文本使用行首匹配符，得到的结果往往是唯一的。例如对一个学生表使用行首匹配符“/^2721010325/”：

```
2721010325…
```

这时得到的结果往往是一个具体且唯一的结果。

4.4.4　行尾匹配符“$”

行尾匹配符“$”用于在文本的每一行的行尾匹配字符串或模式。由于其匹配的是行尾，因此在使用时应当将行尾匹配符放在匹配字符串或模式后面。

（1）例如在文本中匹配所有行尾是 love 的行：

```
/love$/
```

（2）匹配所有只有 5 个字符的行：

```
/^.....$/
```

（3）要匹配所有以 over 结尾的行：

```
/over$/
```

4.4.5　反斜杠屏蔽符“\”

与前面的文件名匹配符一样，反斜杠“\”用来屏蔽一些特殊字符的特殊含义。

由于正则表达式运用在命令中，因此容易被系统误解、在 Shell 中有特殊含义、容易被命令误解的字符都被认为是特殊字符。常见的特殊字符可能是引用符号、注释符号、通配符和逻辑运算符等。下面列举出了一些特殊的符号。

- 注释、分隔符号：#、;等。
- 引用符号：$、``、""等。
- 元字符：.、*、^、$、?、[]、\等。
- 逻辑运算符和操作符：||、&&等。

在命令中使用这些字符时，都应该使用反斜杠将其特殊含义屏蔽。

（1）例如要使用符号“*”做乘法运算，就必须使用反斜杠将其特殊含义屏蔽后再使用：

```
#计算 256 平方
$ expr 256 \* 256
65536
```

（2）例如要使用注释符号“#”和反斜杠“\”：

```
#在命令中输出注释符号和反斜杠
$ echo \#notes\\
#notes\
```

（3）要在正则表达式中表示文件名为“^^.txt”：

```
#在正则表达式中使用含有特殊字符的文件名
/\^\^\.txt/
```

4.4.6　范围匹配符“[]”和排除范围匹配符“[^]”

范围匹配符“[]”与排除范围匹配符“[^]”与文件名通配符中的范围匹配符用法基本相同，也用来匹配几个字符组合，或者是一个字母和数字范围。

（1）例如要匹配单词 love 和 Love：

```
/[Ll]ove/
```

（2）要匹配所有的数字：

```
[0123456789]
```

（3）也可以使用符号“-”表示这个集合：

```
[0-9]
```

（4）而要匹配所有的字母可以使用以下两种方式：

```
[a-Z] [a-z A-Z]
```

（5）不匹配所有的字母：

```
[^a-Z]
```

4.4.7　词首、词尾匹配符“\<”和“\>”

词首匹配符“\<”和词尾匹配符“\>”用于在单词开头和单词结尾匹配特定的字符或模式。虽然词首、词尾匹配符并不常用，但要完整匹配一个单词时，使用词首、词尾匹配符却十分方便。

（1）例如要匹配所有以 disc 为词首的文本，可以使用：

```
/\<disc/
```

（2）要匹配所有以 ment 为词尾的文本，可以使用：

```
/ment\>/
```

（3）要匹配完整的单词 the，可以使用：

```
/\<the\>/
```

4.4.8　重复次数匹配符“x\{\}”

重复次数匹配符“x\{\}”可以全精确匹配字符或字符串连续出现的次数或次数范围。

（1）要匹配字母 m 出现次数为 5 次的文本，可以使用：

```
/m\{5\}/
```

（2）要匹配字母 m 出现次数至少 5 次的文本，可以使用：

```
/m\{5,\}/
```

（3）匹配字母 m 出现次数在 5 至 10 次的文本，可以使用：

```
/m\{5,10\}/
```

4.4.9　组合并使用正则表达式

前几节讲解了正则表达式中常用的基本元字符。但在实际使用时，通常都会将多个元字符组合使用。使用组合之后的匹配模式进行查找，功能更强大、查找更灵活。本小节中将介绍常见的匹配模式。

（1）在查找和筛选信息时，有时需要去除文本中的空行，由于空行没有任何字符，可以使用行首、行尾匹配符匹配空行：

```
/^$/
```

（2）有时一些文本为了看上去更加美观，会使用一种字符填充整行，可以使用以下模式匹配这些行：

```
/^.*$/
```

（3）匹配以 the 或 The 开头的所有行：

```
/^[tT]he/
```

（4）排除以 the 或 The 开头的所有行：

```
/^[^Tt]he/
```

（5）匹配一个 IP 地址：

```
/[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}/
```

（6）匹配一个 6 个字符组成的字符串，前两个字符是字母，中间两个字符是 22 且最后两个字符是小写字母：

```
[a-Z]\{2\}22[a-z]\{2\}
```

（7）要精确匹配单词 love：

```
/\<love\>/
```

上面列举了一些常用的匹配模式，关于这些模式的使用，将在后续章节中详细讲解。

4.5　小　　结

❑ 4.1 节中讲解了 Linux 系统中的引用符号，它们在命令及编写 Bash 脚本时使用得

都十分频繁，因此应该特别注意这些引用符号。

- 4.2 节中介绍了 Linux 系统中的文件名通配符。通过这些文件名通配符，用户可以快速找到需要的文件。
- 4.3 节中讲解了 Bash 中的逻辑运算符和括号。逻辑运算符和括号可以让用户在一行命令中完成比较复杂的功能，经常用于编写精简的脚本，因此初学者应该掌握这些内容。
- 4.4 节中简单介绍了 Linux 系统中的正则表达式、构成正则表达式的基本元字符，以及一些较为常见的匹配模式。正则表达式是 Linux 命令体系中比较重要的内容，在管理和使用系统的过程中经常用到，因此读者应该掌握正则表达式的常见形式，以便需要时能够使用。

Linux 系统中的特殊字符和正则表达式非常重要，在 Shell 编程中应用十分广泛，因此初学者必须了解并掌握这些内容。

第 5 章　查找和筛选工具

Linux 管理员经常需要从众多的命令输出及文本文件中查找和筛选一些信息（例如日志），以便于从中了解服务器的运行状况、系统当前状态等信息。本章将深入讨论如何有效地使用 Linux 下功能强大的工具，从文本或命令输出中查找和筛选出有用的信息。

本章主要涉及的知识点如下。

- 学会如何使用 find 工具查找和处理具有某一类特征的文件。
- 介绍如何使用 grep 工具对输出结果和文本内容进行模式查找。
- 介绍 sed 编辑器，并使用它对多个文件或命令的输出进行重复过滤。
- 学会从格式化报文或文本文件中抽取有价值的信息。
- 学习如何使用 tr 命令转换文本中的字符及删除重复的字符。
- 介绍如何使用多种工具将有价值的文本和内容进行合理的合并和分割。
- 通过每个小节中的实例学习，演示如何从众多的信息中查找和筛选出有价值的信息。

说明： 由于本章中的一些命令从 Unix 中继承而来，因此在不同的系统中使用可能会存在一些微小的差异，用户可以参考相关文档了解这些差异。

5.1　查找文件工具 find

有时可能需要找出一个不知道保存在什么位置的文件，也可能在维护系统的过程中要找出具有某一特征的文件（例如修改时间、文件长度等）。这时可以利用 find 搜索整个文件系统，甚至是网络文件系统上的文件和目录。

与前面学习的文件查找命令相比，find 具有本质的区别，首先，find 是从指定位置进行遍历查找（可以理解为对文件和目录进行逐一查找）。其次，find 可以查找具有某一类特征的文件（例如查找具有某个权限特征的文件等），非常适合于批量处理具有某一类特征的文件。

5.1.1　find 的基本格式

【命令格式】

```
find [path] [expression]
```

【常用选项】

该命令中主要参数如下。

- path：find 查找路径。如果未指定，则默认为当前工作目录。
- Expression：用于定义 find 查找的表达式，表达式通常由选项、测试和动作 3 类参数组成。

选项用于指定 find 查找的目录、帮助等信息，常用的选项及其含义如表 5.1 所示。

表 5.1　常用的选项说明

选　　项	说　　明
help	获得find命令的帮助信息
depth	先从当前目录中查找，然后再从当前目录的子目录中查找
maxdepth *LEVELS*	向下搜索到第*LEVELS*层目录，当*LEVELS*=0时表示只在当前目录查找
mindepth *LEVELS*	至少向下搜索*LEVELS*层目录
mount	不搜索远程文件系统（搜索远程文件系统将花费大量的网络资源）
follow	搜索如果遇到链接文件就连同链接所指向的文件一并检查

大多情况下，Linux 管理员更想搜索到具有某一类特征的文件，这时表达式应当使用一些测试参数，测试参数是一些使得输出更加详细的参数。常用的测试参数及含义如表 5.2 所示。

表 5.2　常用的测试参数说明

测　　试	说　　明
name	按文件名查找
perm	按文件权限查找
type	查找某一类型的文件
mtime +*n*–*n*	按文件修改的时间查找，+*n*表示修改时间距现在*n*天以前，-*n*表示修改时间距现在*n*天以内
atime + *n*–*n*	按文件的访问时间查找文件（使用方法与mtime参数相同）
size *n* [c]	查找文件长度为*n*块的文件，c表示文件大小为*n*字节的文件
User	按文件属主查找
group	按文件属组查找
nouser	查找没有有效属主的文件（文件属主在/etc/passwd文件中不存在）
nogroup	查找没有有效属组的文件（文件属组在/etc/group文件中不存在）

动作参数指定 find 命令如何查找和处理查找到的文件，常用的动作有如下 4 种。

- prune：不在指定目录中查找。
- print：将查找到的文件输出到标准输出。
- exec：对查找到的文件执行 exec 动作后附带的 Shell 命令。
- ok：对查找到的文件执行 ok 动作后附带的 Shell 命令，在每次执行前将提示用户是否执行。

find 有许多参数，大多数情况下，使用该命令都至少要包含一个测试和一个动作，才能完成整个查找任务。

由于本章中的命令参数众多，使用较为复杂，因此本书中将命令的用法示例进行分类讲解。

5.1.2　按文件名称查找

很多时候管理员会忘记某一个文件存放在什么位置。可以使用 name 参数进行查找，也可以使用文件名称通配符配合查找。

（1）想要在/etc 目录下查找 Samba 服务的配置文件 smb.conf：

```
#使用 name 参数指定要查找的文件名，并使用 print 参数将找到的文件输出
# find /etc -name "smb.conf" -print
```

（2）想要查找/etc 目录下所有的配置文件：

```
#使用 name 参数时，配合文件名通配符查找文件
# find /etc -name "*.conf" -print
```

（3）在当前目录中查找名为 messages 的文件：

```
#如果未指定查找的目录，find 将在当前目录中查找
# find -name "messages" -print
```

（4）如果想要在/etc 查找两个小写字母加一个数字，最后面是.d 的文件，可以使用下面这条命令：

```
#查找前两个字符是小写字母，第 3 个字符是数字，后面是.d 的文件
# find /etc -name "[a-z][a-z][0-9].d" -printf
```

（5）在一个很大的文件系统上进行查找时，可能会花费很长时间，通常建议使用操作符&将 find 命令放到后台执行：

```
#使用 find 命令查找文件时，使用操作符&将其放到后台运行
# find / -name "*.conf" -print &
```

5.1.3　按文件权限查找

使用 perm 参数可以按照文件的权限进行查找，使用此参数时，需要使用八进制表示权限。按权限查找文件通常用在多用户系统中，以便于发现可能导致泄密、不安全的内容等。

提示： 使用八进制表示权限的方法称为绝对模式，绝对模式使用数字 4 表示读权限，2 表示写权限，1 表示执行权限，使用多权限时只需要将数值相加即可。使用 3 位数字表示多用户权限，从左起分别表示文件属主、属组和其他用户权限。

（1）在整个文件系统上查找属主可以读、写，属组可以读、写，其他用户可以读的文件：

```
#在整个文件系统上查找权限为 664 的文件
# find / -perm 664 -print
```

（2）在当前目录的 file 子目录下查找其他用户可以读、写、执行的文件（这种情况应当引起重视），此时应该在权限数值前加一个横杠"-"（"-"表示使用包含模式）：

```
#在当前目录的 file 子目录中查找权限中包含其他用户可读、写、执行的文件
# find ./file -perm -007 -print
```

上述命令将输出其他用户可以读、写、执行的所有文件，属主及属组的访问权限将被 find 忽略。

5.1.4　按文件类型查找

用 type 参数可以按类型查找文件，常见的文件类型有目录、字符设备文件和普通文件等。按文件类型进行查找适合于一些比较特殊的情况，例如要查找某个设备而不知道其具体的名称等。

（1）查找/dev 下有哪些字符设备：

```
#查找/dev 目录中的字符设备，其中 c 表示字符设备
# find /dev -type c -print
/dev/vcs5
/dev/vcsa5
/dev/vcsa3
……
```

此时 find 列出了目录/dev 中的所有字符设备，其中包括光驱、终端、控制台等。

（2）查找目录/dev 中的块设备文件：

```
#查找目录/dev 中的块设备文件，其中 b 表示块设备文件
# find /dev -type b -print
/dev/md0
/dev/fd0u800
/dev/fd0u1120
……
```

此时 find 列出了目录/dev 中的所有块设备文件，包括磁盘、软驱、阵列等。

（3）查找目录/etc/rc3.d 中除了链接文件以外的文件（通常情况下这个目录下只会存放链接文件）：

```
#查找目录/etc/rc3.d 中除了链接文件之外的文件
#其中 l 表示链接文件，感叹号!表示否定
# find /etc/rc3.d ! -type l -print
```

提示： 参数 type 使用的文件标识除普通文件使用 f 作为标识外，其他标识都与 ls 命令的长格式中的文件类型标识一致，例如使用字母 d 表示目录，l 表示链接文件等。

5.1.5　按文件的时间戳记和大小查找

在管理和维护 Linux 系统的过程中，经常需要清理一些过期的日志和文件，清理这些文件的标准通常是文件的修改时间、访问时间、文件的大小等。而另一种情况是我们需要知道一定时间内被修改过的文件，以便对这些文件进行备份等。这时可以使用 find 命令的

mtime、atime 和 size 参数按文件的时间戳记和文件的长度查找。

1．按时间戳记查找文件

按文件的时间戳记进行查找时，可以使用+*n* 限定时间在 *n* 天以前，使用-*n* 限定时间在 *n* 天以内。虽然也可以使用 *n* 精确限定时间，但一般不这样使用，原因是系统计算时间是以秒为计算单位的，因此一般情况下该选项无法得到任何有用的结果。

（1）希望在整个文件系统上查找修改时间在一周以内的文件：

```
#使用-7 指定修改时间在 7 天以内的文件
# find / -mtime -7 -print
```

（2）查找用户根目录下修改时间在 1 天以前的文件：

```
#使用+1 指定修改时间在 1 天以前的文件
# find ~ -mtime +1 -print
```

（3）查找目录/data 中访问时间在 10 天以内的文件：

```
#使用 atime -10 指定访问时间在 10 天以内的文件
# find /data -atime -10 -print
```

2．按长度查找文件

使用 size 按文件大小查找时，可以像按时间戳记那样使用+*n* 表示文件长度大于 *n* 的文件，-*n* 表示文件长度小于 *n* 的文件。默认情况下文件长度的单位是块（一块等于 512 字节），如果要以字节来计算文件长度，应该在数字后加小写字母 c。

（1）在当前目录下查找文件长度大于 10MB 的文件：

```
#使用+10000000c 表示文件长度大于 10MB 的文件
# find . -size +10000000c -print
```

（2）要在当前目录下查找文件长度小于 30 块的文件：

```
#使用-30 表示文件长度小于 30 块（512*30=15KB）的文件
# find . -size -30 -print
```

在按文件大小查找文件时，也可以使用 KB（千字节）、MB（兆字节）、GB（吉字节）等较大的单位标志文件的大小。

5.1.6　按文件属主或属组查找

当管理员将一个用户或用户组从系统中删除时，可能需要将该用户或用户组的文件收集起来保存一段时间。这时可以使用 find 命令的 user、nouser、group 和 nogroup 这几个参数，查找指定用户、用户组的文件。

（1）要查找目录/home 中属主为 lilei 的文件：

```
#使用 user 参数查找属主为 lilei 的文件
# find /home -user lilei -print
```

（2）在整个文件系统上查找出没有有效属主（即用户名在系统用户名/etc/passwd 中不存在）的文件：

```
#使用 nouser 参数查找没有有效属主的文件
# find / -nouser -print
```

（3）查找目录/file 中属组为 admin 的文件：

```
#使用 group 参数查找属组为 admin 的文件
# find /file -group admin -print
```

（4）在整个文件系统中查找没有有效属组（即用户组名称在系统用户组文件/etc/group 中不存在）的文件：

```
#使用 nogroup 参数查找没有有效属组的文件
# find / -nogroup -print
```

5.1.7　find 工具的其他参数

在 find 命令中还存在一些其他参数，这些参数使用频率较低（例如忽略某个目录），但有时使用这些参数查找文件却十分方便。

（1）忽略目录参数 prune

在查找文件的时候，可能不需要查找某个目录，或用户已经知道某个目录中不存在这个文件，这时就可以使用 prune 参数忽略这些目录。

例如要从除了/etc 以外的整个文件系统上查找以.conf 结尾的文件：

```
#使用 path、prune 和 name 参数指定在除了/etc 以外的整个目录中查找以.conf 结尾的配置文件
# find / -path "/etc" -prune -o -name "*.conf" -print
```

上面的示例中，使用参数 o 将两个不同的参数连接起来。

（2）忽略远程文件系统参数 mount

在整个文件系统上查找文件时，如果系统上挂载有远程文件系统，搜索远程文件系统不仅要耗费大量网络资源，还要花费大量时间。这时可以使用 mount 参数忽略挂载的远程文件系统。

例如要在本地文件系统上查找一个名为 file 的文件：

```
#使用 mount 参数忽略远程文件系统
# find / -name "file" -mount -print
```

注意：使用 prune 参数查找文件时，如果同时使用了 depth 参数，find 会将 prune 参数忽略。

5.1.8　使用 exec 和 ok 处理查找到的文件

在 Linux 管理和维护中，大多数时候查找具有某一类特征的文件的目的是为了处理这些文件，通常的处理方式有删除、移动等，例如查找并删除一些旧的文件或过期的日志等，这时可以使用动作参数中的 exec、ok，这两个参数都可以对查找到的文件执行 Shell 命令。

不同的是，使用 ok 参数执行较危险的 Shell 命令（例如删除文件命令 rm）时会提示用户。

利用 find 命令查找到文件后，就可以使用 exec、ok 参数对查找到的文件执行 Shell 命令。使用 exec、ok 参数执行 Shell 命令的格式如下：

```
-exec [Shell 命令] {} \;
-ok [Shell 命令] {} \;
```

在上面的格式中，参数 exec、ok 后面空一格紧跟要执行的 Shell 命令，再空一格后面是一个大括号“{}”，最后加上一个反斜杠“\”和一个分号“;”。

（1）查找当前目录的 backup_sys 子目录中，修改时间距现在一周以前、以 message 开头的文件，并用 ls 命令查看：

```
#在当前目录的 backup_sys 子目录中，查找修改时间在一周以前、以 message 开头的文件
#然后将找到的文件交给 ls -l 命令
# find ./backup_sys -name "message*" -mtime +7 -exec ls -l {} \;
-rw-r--r-- 1 root root 4095 Jul 11 04:00 ./backup_sys/message_backup_20100711
-rw-r--r-- 1 root root 7233 Jul 18 04:00 ./backup_sys/message_backup_20100718
```

从命令的结果中可以看出，ls 命令列出的文件是相对于当前目录的相对路径。

（2）查找并删除两周以前的备份文件：

```
#查找修改时间在两周以前的文件，并使用 exec 参数将找到的文件交给 rm 命令删除
# find ./backup_sys -name "message*" -mtime +14 -exec rm {} \;
```

执行上述命令后，可以使用 ls 命令查看，文件 message_backup_20100711 被删除，但是上面这个命令执行过程中没有任何提示信息。

（3）如果要在执行 Shell 命令时获得命令执行的提示，可以使用 ok 参数。例如删除两周以前的备份文件并在执行前获得提示：

```
#使用 ok 参数执行较危险的 Shell 命令时将获得提示信息
# find ./backup_sys -name "message*" -mtime +14 -ok rm {} \;
#以下为提示信息
< rm ... ./backup_sys/message_backup_20100711 > ? y
```

5.1.9　使用 xargs 命令处理查找到的文件

利用 exec 和 ok 参数处理查找到的文件时，存在一些缺陷，这些缺陷如下。

- 系统对参数 exec、ok 传递给 Shell 命令的文件列表长度有一定的限制。当 find 命令查找到的文件数量很多时，会出现参数列表溢出错误。
- 参数对 find 命令找到的每一个文件发起一个相应的处理进程，当 find 命令查找到的文件数量很多时，可能会影响整个系统性能。

xargs 命令的作用是构造一个参数列表并交给命令执行。与参数 exec、ok 相比，xargs 不会一次获取并处理 find 找到的所有文件，而是每次只获取并处理其中的一部分。处理完后再获取下一部分，直至结束。整个过程 xargs 都只发起一个处理进程，对系统性能的影响很小。

使用 xargs 命令分割参数列表时，需要借助于管道。例如查找当前目录的 backup_sys 子目录中，修改时间距现在两周以前的所有文件并删除：

```
#使用管道将找到的文件列表交给 xargs 命令分割之后使用 rm 命令删除
# find ./backup_sys -name "message*" -mtime +14 -print | xargs rm
```

这条命令首先使用 find 查找出相应的文件，然后使用管道传递给 xargs，xargs 命令构造参数列表之后再传递给 rm 命令执行删除操作。

注意：无论使用怎样的方式处理查找到的文件，在执行删除和移动等命令时，一定要先确认之后再删除。

5.1.10　find 工具应用实例

在前面几个小节中，讲解了如何利用 find 工具查找文件和目录，本小节将利用几个简单的例子讲解 find 工具的实际应用。

1．使用find查找需要备份的文件

在 Windows 中，系统会给每个文件加上存档标记，每次使用完全备份和差异备份后，备份工具会将文件的存档标记清除。当文件被修改或创建时，系统又会自动给文件加上存档标记，因此备份软件只需要备份被标记的文件即可实现增量备份。但在 Linux 系统中文件没有存档标记，此时可以利用文件被创建或修改时，Linux 系统给文件加上或更新的时间戳记收集需要备份的文件。

在这个例子中，管理员在每周周一的凌晨 4 点对/file 目录进行完全备份，完全备份时使用归档命令 tar（tar 命令将在第 9 章中介绍）：

```
#使用 tar 命令备份目录/file，并将备份文件命名为 file_backup_20100725.tgz
# tar -czf /backup/file_backup_20100725.tgz /file
```

在第二天的凌晨 4 点，管理员首先使用 find 命令统计在过去的这一天内哪些文件被修改过，然后使用 tar 命令对修改过的文件进行备份：

```
#使用 find 命令查找一天以内修改过的普通文件，并将找到的文件存入文件/backup/file_list 中
# find /file -type f -mtime -1 -print >/backup/file_list
#使用 tar 命令备份/backup/file_list 中记录的文件
# tar -czT /backup/file_list -f /backup/file_backup_20100726.tgz
#删除使用过的文件列表
# rm -rf /backup/file_list
```

提示：由于目录中的文件被修改后，目录本身的时间戳记也会发生变化，因此上面这个示例中，使用 find 命令查找的对象是普通文件（而不是所有文件）。

2．删除过期文件

在备份任务进行了一段时间之后，存储的备份文件将会占用大量的存储空间。为了节约存储空间，应该及时将过期的备份文件删除。

在这个示例中，每个工作周的第一天，利用 find 命令查找出两周以前的备份文件，然后将其删除。删除文件前，应该先查看并确定要删除的文件：

```
#使用 find 命令查看修改时间为 14 天以前的文件，并使用 ls -l 命令查看
# find . -mtime +14 -exec ls -l {} \;
-rw-r--r-- 1 root root 45623189 Jul  5 04:00 ./file_backup_20100705_full.tgz
-rw-r--r-- 1 root root   546478 Jul  8 04:00 ./file_backup_20100708.tgz
-rw-r--r-- 1 root root   321554 Jul  9 04:00 ./file_backup_20100709.tgz
……
```

确认没有问题之后，就可以删除这些过期的文件：

```
#确认无误后，将找到的文件删除
# find . -mtime +14 -exec rm {} \;
```

5.2　查找文本工具 grep

许多时候需要从一大堆的命令输出或文本文件内容中找出一两行关键的内容，例如从系统用户文件中查找某个用户。如果不借助工具，这将是一项十分繁琐的工作，这时可以使用 grep 工具对内容进行筛选。

grep 是 global regular expression print（全局正则表达式打印）的缩写，它来源于最早的行编辑器 ed。grep 是管理和维护系统时经常用到的一个工具，在前几章中已有该命令的应用实例，本节将继续使用实例讲解 grep 工具的应用。

5.2.1　grep 的基本格式

【命令格式】

```
grep [option] pattern [file]
```

grep 工具在文件 file 中查找与字符串 pattern 匹配的内容，如果找到，则将整行输出到标准输出。

【常用选项】

- i：忽略大小写。
- n：将结果输出的同时，也输出该行的行号。
- s：在没有查找到匹配的内容时，不显示错误信息。
- l：从多个文件中查找时，只输出找到匹配内容的文件名称。
- h：从多个文件中查找时，只输出匹配的内容，不显示文件名称。
- c：只输出匹配内容的总行数。
- v：反转查找，即输出匹配内容以外的行。

grep 工作时，总是以行为单位查找。首先将文本的第 1 行读入缓冲区并执行查找，如果找到匹配的字符串，则输出整行。否则就丢弃缓冲区内容并读入下一个文本行继续查找，直到文本结束。

5.2.2　使用 grep 查找文本

在开始讲解之前，引入一个示例文本文件 students。students 文件中包含的是一些学生信息，这个文件的结构如下。

- 第 1 列：学号（入学年份+学院编号+专业编号+班级号+班级学号）。28 代表 2008 年入学，其他依此类推。
- 第 2 列：学生姓名。
- 第 3 列：学生所在省份。
- 第 4 列：辅导员姓名。
- 第 5 列：学生的出生年月。
- 第 6 列：高等数学成绩（公共课）。
- 第 7 列：英语成绩（公共课）。
- 第 8 列：计算机基础成绩（公共课）。
- 第 9 列：线性代数成绩。
- 第 10 列：总分。
- 第 11 列：平均分。

查看 students 的内容：

```
#使用 cat 命令查看文件 students 的内容
# cat students
2821020225 Liulu    Sichuan Lixia   01/23/93    89  76  88  72  325 81
2821020115 Liumin   Henan   lixia   05/14/94    78  65  59  78  280 70
2721010321 Xuli     Jiangsu Luolei  12/25/92    76  81  85  79  321 80
2921020632 Xiayu    Shanxi  Hetao   03/26/93    78  86  92  78  334 84
2721010409 Liwei    Sichuan tangwei 11/21/92    98  88  85  85  356 89
2921050313 Heli     Xizang  Tangwei 07/12/94    56  78  80  45  259 65
2721030227 Wangtao Yunnan  Huli    03/21/93    87  76  69  88  320 80
```

（1）查找关键字

与使用搜索引擎一样，使用 grep 命令查找时，需要提供一个关键字。为了保证查找的效率，应该尽量选择一个“独一无二”的关键字，以避免查找到无关的信息。

例如要在 students 文件中查找字符串 92：

```
#在 students 中查找包含 92 的行
# grep "92" students
2721010321 Xuli     Jiangsu Luolei  12/25/92    76  81  85  79  321  80
2921020632 Xiayu    Shanxi  Hetao   03/26/93    78  86  92  78  334  84
2721010409 Liwei    Sichuan tangwei 09/21/92    98  88  85  85  356  89
2921050313 Heli     Xizang  Tangwei 07/12/94    56  78  80  45  259  65
```

grep 输出了文件中含有字符串 92 的 3 行内容。

（2）显示行号

有时可能需要确定查找到的结果处于文本的什么位置，这时可以配合选项 n 在查找时输出行号。

输出查找结果的同时输出行号：

```
#查找所有包含 92 的行，并显示该行的行号
# grep -n "92" students
3:2721010321    Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
4:2921020632    Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
5:2721010409    Liwei   Sichuan tangwei 09/21/92  98  88  85  85  356  89
6:2921050313    Heli    Xizang  Tangwei 07/12/94  56  78  80  45  259  65
```

（3）统计结果

有时查找到了许多结果，但我们却只想知道一共有多少结果，而不是具体内容。例如需要统计来自某一地区的学生数量，某个产品一共有多少条出货记录等。这时可以配合使用选项 c 不显示查找结果，仅输出查找到的结果条数。

统计文件中一共有多少名来自西藏的学生：

```
#使用 c 选项统计查找到的总行数
# grep -c "Xizang" students
1
```

（4）大小写敏感

grep 命令继承了 Linux 系统对大小写字母敏感的特性，因此如果要使 grep 在查询过程中不区分大小写，应该配合使用 i 选项。

在 students 文件中有两名辅导员的姓名首字母没有大写，查询哪些学生的辅导员是 Tangwei：

```
#使用选项 i 查找时忽略大小写
# grep -i "tangwei" students
2721010409  Liwei   Sichuan tangwei 09/21/92  98  88  85  85  356  89
2921050313  Heli    Xizang  Tangwei 07/12/94  56  78  80  45  259  65
```

（5）反转查找

有时需要进行反转查找，即显示那些不匹配字符串的行，此时可以配合选项 v。例如想要查询辅导员不是 Tangwei 和 Lixia 的学生情况：

```
#使用选项 v 反转查找不含有 tangwei 和 lixia 的行
#为了满足两个条件，使用两次查找
# grep -vi "tangwei" students | grep -vi "lixia"
2921020632  iayu    Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2721030227  angtao  Yunnan  Huli    03/21/93  87  76  69  88  320  80
```

这条命令中的第 1 个 grep 先输出不含有 Tangwei 的所有行，然后通过管道将结果传递给第 2 个 grep 命令，第 2 个命令输出不含有 Lixia 的所有行。

（6）多文件查找

有时需要从多个文件中查询一些相关联的内容，这时就要用到多文件查询。例如管理员要从目录/etc 的文件中查找有关 root 用户的内容：

```
#显示/etc 目录中所有包含有 root 的文件名
# grep -l "root" /etc/*
/etc/aliases
/etc/aliases.db
/etc/anacrontab
```

```
/etc/crontab
……
```

grep 输出了目录/etc 中所有含有 root 字符串的文件名。

查询密码文件/etc/passwd 和影子文件/etc/shadow 中含有字符串 root 的所有行，并且不显示文件名称：

```
#使用 h 选项显示系统用户文件和影子文件中包含 root 的行
# grep -h "root" /etc/passwd /etc/shadow
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
root:$1$7xRyz4dM$6J4SA75DYrMEyA2zLnFiC/:14816:0:99999:7:::
```

（7）在命令输出和变量中查找

grep 不仅可以从文件中查询字符串，还可以从字符串和字符串变量中查询：

```
#使用 grep 在命令输出中查找
# echo "Welcome to Beijin" | grep "to Beijin"
Welcome to Beijin
#使用 grep 在变量中查找
# A="Welcome to Beijin"
# echo $A | grep "Beijin"
Welcome to Beijin
#使用变量保存查找的字符串
# A="Beijin"
# echo "Welcome to Beijin" | grep "$A"
Welcome to Beijin
```

注意：在使用 grep 查找时，被查找的字符串可以不使用引号。但在被查找的字符串中有空格或被查询的字符串保存在一个变量中时，应该使用引号以免被 grep 误解为一个命令或参数。

5.2.3　行首、行尾匹配查找

文本的行首和行尾通常用于保存特殊意义的字段，例如产品序号、销售额等内容，因此从行首和行尾匹配查找可能会比较频繁。

1．行首匹配

一些格式化文本通常将关键信息放在行首，例如学号、货号、编码等，因此如果在行首进行查找，通常都可以得到更精确的内容。

查询文件 students 中所有 2008 年入学的学生：

```
#使用行首匹配符显示所有行首字符串为 28 的内容
# grep '^28' students
2821020225  iulu    Sichuan Lixia      01/23/93  89  76  88  72  325  81
2821020115  iumin   Henan   lixia      05/14/94  78  65  59  78  280  70
```

使用行首匹配符时，也可以与其他正则表达式元字符配合使用，例如查询所有专业编号为 02 的学生：

```
#使用行首匹配符显示第 5、6 个字符为 02 的所有行
# grep '^....02' students
2821020225  iulu    Sichuan Lixia   01/23/93  89  76  88  72  325  81
2821020115  iumin   Henan   lixia   05/14/94  78  65  59  78  280  70
2921020632  iayu    Shanxi  Hetao   03/26/93  78  86  92  78  334  84
```

2. 行尾匹配

格式化文本的行尾通常为一些统计信息，例如销售总计、商品总价、平均成绩等，因此在行尾查找通常可以筛选出更有用的内容。

例如要查找学院优秀生（平均成绩 85 以上）：

```
#使用选项 v 反转显示行尾大于 85 的行
# grep -v '[0-7][0-9]$' students | grep -v '8[0-4]$'
2721010409  iwei    Sichuan tangwei 11/21/92  98  88  85  85  356  89
```

上面这个示例相对较复杂，第 1 个 grep 用来剔除平均成绩在 80 分以下的学生，然后通过管道将结果传递给第 2 个 grep 命令，第 2 个 grep 命令则剔除平均成绩在 80～84 分之间的学生。

5.2.4　配合常用的正则表达式查找

除了上一小节中介绍的行首和行尾外，有时也会使用其他的正则表达式。本小节将简单介绍 grep 中常用的正则表达式。

（1）使用范围模式匹配

在使用范围匹配符时，并不一定非要从开头到结尾，也可以是从中间某个特定的字符开始到某个特定的字符结束。例如查询所有 93 年以后出生的学生的详细情况：

```
#使用范围匹配符查找所有 93 年之后出生的学生
#使用/9[3-9]这样的方式避免匹配到成绩
# grep '/9[3-9]' students
2821020225  iulu    Sichuan Lixia   01/23/93  89  76  88  72  325  81
2821020115  iumin   Henan   lixia   05/14/94  78  65  59  78  280  70
2921020632  iayu    Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2921050313  eli Xizang  Tangwei 07/12/94  56  78  80  45  259  65
2721030227  angtao  Yunnan  Huli    03/21/93  87  76  69  88  320  80
```

（2）次数匹配模式

有时需要知道一个匹配模式出现的次数，为了讲解其使用方法，此处引入另一个示例文件 file.txt：

```
#使用 cat 命令查看示例文件 file.txt 的内容
# cat file.txt
sleep
sleepdddddd
bbbbbbbbd

ccccd
```

查找字母 c 至少重复出现 2 次的所有行：

```
#使用次数匹配模式查找字母 c 至少出现 2 次的行
# grep 'c\{3,\}' file.txt
ccccd
```

查找字母 e 重复出现 2 次且以 p 结尾的行：

```
#使用次数匹配时配合其他字符
# grep 'e\{2\}p$' file.txt
sleep
```

查找字母 b 重复出现 2 到 10 次的：

```
#使用次数范围匹配
# grep 'b\{2,10\}' file.txt
bbbbbbbbd
```

查找文件“file.txt”中有多少个空行：

```
#使用行首、行尾匹配表示空行并使用 c 选项统计
# grep -c '^$' file.txt
1
```

（3）处理特殊字符

在使用 grep 查找的过程中，可能会遇到一些特殊符号，可以使用反斜线“\”屏蔽掉这些特殊字符的含义。

例如要查看当前目录下所有以.txt 结尾的文件：

```
#使用反斜线屏蔽掉点号的特殊含义
# ls -l | grep '\.txt$'
-rw-r--r-- 1 root root 1211 Jul 26 14:00 file1.txt
-rw-r--r-- 1 root root 2144 Jul 26 14:01 file2.txt
-rw-r--r-- 1 root root   34 Jul 26 13:33 file.txt
```

注意：在结合正则表达式进行查询时，正则表达式可以不使用单引号。但为了避免被 grep 误解，最好将正则表达式放在单引号内。

5.2.5　使用或、与多匹配模式查找

有时需要进行多匹配模式查找，即在同一条命令中使用多个匹配模式。多匹配模式的关系可以是或匹配模式（即只需满足一个匹配模式即可），也可以是与匹配模式（即需要满足所有匹配模式）。

（1）或匹配模式

使用参数 E 让 grep 命令将要匹配的字符串延伸为一个普通的表达式，此时可以使用竖线“|”表示或匹配模式，即只需要匹配两个字符串中的任意一个即可。

例如查询来自河南和云南学生的详细情况：

```
# grep -E 'Henan|Yunnan' students
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  280  70
```

```
2721030227  Wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
```

（2）与匹配模式

然而参数 E 并不支持与匹配模式查询，此时可以使用多条件管道实现。例如要查询 Lixia 的学生中有哪些来自 Sichuan：

```
# grep -i "Lixia" students | grep "Sichuan"
2821020225  Liulu   Sichuan Lixia       01/23/93  89  76  88  72  325  81
```

5.2.6　grep 工具应用实例

在管理和维护 Linux 的过程中，查找文本工具 grep 使用频率非常高，因此建议初学者掌握这个命令。本小节将用几个示例简单介绍 grep 的实际应用。

1．精简配置文件

在 Linux 系统中，管理员经常接触到各种类型的服务配置文件。这些配置文件通常都使用了一个通用的注释格式，即使用井号“#”（通常是注释信息）或“;”（通常标志该行是默认设置）作为开头标志。配置文件中的注释信息和默认配置语句行写得非常详细，这些语句行比真正起作用的配置往往多出十数倍甚至更多。

通常情况下，熟悉这些配置文件的管理员会使用 grep 工具的参数 v 精简这些配置文件，让这些配置文件的可读性更高，更方便修改。此处以精简 Samba 服务的配置文件 smb.conf 为例：

```
#进入配置文件目录并使用 mv 命令备份配置文件
# cd /etc/samba/
# mv smb.conf smb.conf_backup
#使用反转查找去掉包含井号、分号的行和空行，最后使用重定向生成新的配置文件
# cat smb.conf_backup | grep -v '#' | grep -v '^;' | grep -v '^$' >smb.conf
#使用 cat 命令查看生成的配置文件
# cat smb.conf
[global]
        workgroup = MYGROUP
        server string = Samba Server Version %v
        security = user
        passdb backend = tdbsam
……
```

使用 cat 命令查看精简之后的配置文件 smb.conf，原本有 288 行的配置文件经过精简，现在仅剩 18 行。

2．从系统管理命令输出中查找

grep 工具是 Linux 中最常用的命令之一，管理员需要经常使用它对命令的结果执行筛选，以便于查看关键内容。

例如从服务列表中筛选出蓝牙服务，以便于查看这个服务在不同运行级别中的启动状态：

```
#先使用 chkconfig 命令列出所有服务列表，然后使用 grep 筛选蓝牙服务 bluetooth
# chkconfig --list | grep bluetooth
```

```
bluetooth       0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

说明：grep 工具有一个很庞大的家族，除了在本节中讲到的标准的 grep 外，还有 egrep（扩展 grep）、fgrep（快速 grep）和 agrep（近似 grep）等。感兴趣的读者可以参考相关帮助手册来了解这些 grep 变体。

5.3　流编辑器 sed

Sed 与 grep 一样，都起源于老式的 ed 编辑器，因其是一个流编辑器（stream editor）而得名。与 Vi 等编辑器不同，sed 是一种非交互式编辑器（即用户不必参与编辑过程），它使用预先设定好的编辑指令对输入的文本进行编辑，完成之后再输出编辑结果。

sed 工作时，首先读取文本中的第 1 行，将其放入一个被称为模式空间的临时缓冲区内。然后再读取第 1 条编辑指令，使用指令中定义的模式和行号查找、编辑文本。完成编辑后，将结果输出并读取下一行，重复这个过程直到文本结束。除了模式空间之外，sed 还使用了一个被称为保留空间的临时缓冲区，保留空间通常用于暂存编辑内容。

sed 通常用来对多个文件或命令输出进行重复处理，以达到简化操作的目的。它是一个非常重要的文本过滤工具，在编写系统维护和管理的脚本时经常用到。本节将通过示例详细讲解 sed 的各种用法。

5.3.1　sed 基本格式

【命令格式】

```
sed [option] command input-file
sed [option] -f script-file input-file
```

在上面的命令格式中，展示了调用 sed 命令的两种方法。在第 1 种调用方法中，将编辑指令直接放在选项后面，这是最为常见的一种。当执行一些较为复杂的编辑操作时，可能使用的编辑指令会很长，这时为了便于阅读，通常将编辑指令放入一个脚本文件中，通过第 2 种方法来调用 sed 编辑文本。

【常用选项】

选项用于指定编辑行为、打印设置等，常用的选项如下。

- n：不输出所有行，默认情况是输出所有行。
- e：允许在该选项后面加一条新的编辑指令。当有多条编辑指令时，应该使用该选项逐一添加，如果编辑指令只有一条，可以不使用该选项。
- f：用于指定装有编辑指令的脚本文件。
- h：输出 sed 的帮助信息。

【常用的定位方式】

使用 sed 编辑文本时，应该向其指出要编辑的位置信息，通常使用一个行号或指定一个行号范围，也可以使用正则表达式对要编辑的文本进行模式匹配。常见的操作如下。

- n：表示行号为 *n* 的行。
- m,n：表示一个行号的范围，从第 *m* 行到第 *n* 行。

- m,n!：排除第 *m* 行到第 *n* 行。
- /pattern/：表示匹配 pattern 的所有行。
- /pattern1/pattern2/：表示匹配 pattern1 和 pattern2 的所有行（需要使用选项 e 将两个模式隔开）。
- /pattern/,~n：从匹配 pattern 的行开始，向后的 *n* 行之间的所有行（即包括匹配行在内，一共 *n* 行）。
- /pattern/,+*n*：从匹配 pattern 的行开始，向后的 *n* 行的所有行（即包括匹配行在内，一共 *n*+1 行）。
- n,/pattern/：表示从第 *n* 行开始到匹配 pattern 之内的所有行。

【常用的编辑指令】

使用 sed 编辑文本时，必须要使用对文本进行操作的指令（称为编辑指令），使用编辑指令可以对文本执行修改、删除和替换等操作。常用的编辑指令及说明如表 5.3 所示。

表 5.3　常用的编辑指令及说明

常用的编辑指令	说　　明
p	将指定的行输出到标准输出
P	将模式空间中第1个换行符“\n”之前的文本输出到标准输出（选项大写）
=	输出匹配模式所在的行号
d	移除模式空间中的行并读取下一行
D	删除模式空间中最后一个换行符之前的文本。如果模式空间为空，则将下一行文本读取到模式空间中
a*text*	在指定行后面加入新的文本信息*text*
i*text*	在指定行前面加入新的文本信息*text*
c*text*	使用新文本信息*text*替换定位行
r *file*	从文件*file*中读取文本
w *file*	将结果写入*file*文件中
s/*pattern1*/*pattern2*/flag	将与模式*pattern1*相匹配的文本用*pattern2*替换。默认情况下，只替换第1次匹配到的内容。指令s之后的/分隔符可以使用任意字符代替
n	如果没有禁止默认输出所有行功能（即没有使用选项n），则将模式空间的内容输出到标准输出。如果禁止默认输出功能，就读取下一行文本并替换模式空间中的文本
N	在当前模式空间中的文本末尾加上换行符\n，并读取下一行文本追加到换行符之后
! *command*	对没有被定位的行使用!之后的*command*编辑指令
y/*pattern1*/*pattern2*/	将与模式*pattern1*相匹配的内容用*pattern2*替换，*pattern1*和*pattern2*的字符长度应该相等
l	将模式空间的文本和控制字符以肉眼可见的方式输出到标准输出
{*sub_command*}	执行大括号内的子命令或子命令组
g	清除模式空间并将保留空间的文本放进模式空间（即模式空间<保留空间）
G	将保留空间的文本追加到模式空间的文本之后（即模式空间<<保留空间）
h	清除保留空间并将模式空间的文本放入保留空间（即模式空间>保留空间）
H	将模式空间的文本追加到保留空间的文本之后（即模式空间>>保留空间）
x	在保留空间的文本和模式空间的文本进行互换
Q	退出指令。使用q指令可以让sed立即退出，不再执行后面的操作

在上面的编辑指令中，替换匹配模式指令 s 的 flag 选项通常向 sed 指示如何替换或替换成功后要进行的操作。常见的 flag 选项如下。

- g：将所有匹配到的内容全部替换。
- n：仅对行内第 *n* 次匹配到的内容进行替换。
- p：如果进行了替换，就将模式空间的文本输出到标准输出。
- w *file*：如果进行了替换，就将模式空间的文本写入文件 *file* 中。

一个编辑指令只能对指定位置进行一次编辑操作，如果要对指定位置执行多次编辑操作，可以使用大括号“{}”将多条编辑指令放入其中。而对多个位置进行多次编辑可以使用 e 选项添加多条编辑指令，也可以使用管道并配合多个 sed 命令。

【常用的分支语句】

除了前面介绍的编辑指令外，sed 还为执行一些复编辑操作提供了分支语句（有点类似于 C 语言中的 if 和 goto 语句）。常用的分支语句有以下几种。

- :label：标记点。标记一个位置以便于跳转语句跳转到指定位置。可以使用任何 8 个或少于 8 个字符的字符串来做标记点。
- t label：测试指令，如果上一条替换指令成功执行，则转移到标记点 label 处继续执行。如果没有 label，则转移到末尾。
- b label：无条件转移到标记点 label 处继续执行。如果没有 label，则转移到末尾。
- #：一般用于脚本的注释。将#号后面的内容视为注释，sed 将忽略后面的内容，直接执行下一行。如果出现在脚本的第 1 行，且后面是“n”（即“#n”），sed 将不禁止默认地输出所有行。

使用了分支语句的编辑指令通常都很长，一般将这些指令放在脚本文件中，然后使用 f 选项调用 sed。

sed 有许多种用法，想学好它并不难，从小的命令和脚本开始着手多加练习，很快就可以掌握这个工具的用法。

注意： 使用 sed 时，为了便于阅读，通常将编辑指令放在单引号或双引号中。在有些系统中，只能使用单引号和双引号其中的一种，读者需要特别注意。

5.3.2　显示和删除行

在 5.2 节中简单介绍了查找文本命令 grep，利用显示和删除行命令 sed 也可以实现 grep 的查找功能。本小节将使用 5.2 节中的示例文件介绍如何使用 sed 显示、删除行。

sed 命令执行结束后，通常将结果输出到屏幕上，如果需要保存 sed 命令的输出结果，可以使用输出重向的方法。

说明： 本书的示例中，通常将编辑指令放入单引号中，以便于阅读及屏蔽某些特殊字符（例如屏蔽变量引用符“$”）。

1. 显示行

显示行需要使用 sed 的编辑指令 p，指定显示内容可以使用行号、行号范围、正则表

达式等。

（1）使用数字 3 指定输出文本的第 3 行：

```
#使用数字 3 指定输出的行号
#由于 sed 默认输出所有行，因此使用选项 n 屏蔽掉默认的输出
# sed -n '3p' students
2721010321  Xuli    Jiangsu  Luolei   12/25/92  76  81  85  79  321  80
```

（2）输出指定的行时，也可以指定一个输出的范围，例如要输出第 3 行到第 5 行：

```
#指定输出文本的第 3 行到第 5 行
# sed -n '3,5p' students
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2721010409  Liwei   Sichuan tangwei 11/21/92  98  88  85  85  356  89
```

（3）大多数时候可能我们并不知道要输出的行的具体行号，此时可以使用模式匹配的方式定位要输出的行。例如要输出所有辅导员是 Tangwei 的学生情况：

```
#使用模式[tTangwei]定位操作行
# sed -n '/[tT]angwei/p' students
2721010409  Liwei   Sichuan tangwei 11/21/92  98  88  85  85  356  89
2921050313  Heli    Xizang  Tangwei 07/12/94  56  78  80  45  259  65
```

（4）可以同时使用行号和模式匹配来指定要输出的行。例如输出从第 2 行开始到匹配含有 Hetao 的行之间的所有文本：

```
#同时使用行号和匹配模式指定操作行的范围
# sed -n '2,/Hetao/p' students
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  280  70
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
```

（5）可以使用行首或行尾匹配来指定要输出的行。例如要查找所有 2009 年入学的学生情况：

```
#使用行首匹配输出所有 2009 年入学的学生
# sed -n '/^29/p' students
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2921050313  Heli    Xizang  Tangwei 07/12/94  56  78  80  45  259  65
```

（6）如果要查找某个匹配行的位置，可以不输出匹配的行，只输出匹配的行号。例如要输出西藏的学生在哪些行：

```
#使用编辑指令=输出匹配位置的行号
# sed -n '/Xizang/=' students
6
```

（7）也可以匹配一行，然后输出匹配行下面 *n* 行之内的所有行。例如输出含有 Luolei 的行和其后的 1 行内容：

```
#显示从匹配模式/Luolei/开始向下的 1 行
# sed -n '/Luolei/,~2p' students
```

```
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
```

2．删除行

许多时候我们希望能够删除文本中的内容（类似于 grep 命令中的反转查找），此时可以使用编辑指令 d 删除匹配的行。

（1）例如要删除所有 2008 年入学的学生信息：

```
#由于指令 d 会删除模式空间中的内容，因此可以不使用选项 n
# sed '/^28/d' students
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2721010409  Liwei   Sichuan tangwei 11/21/92  98  88  85  85  356  89
2921050313  Heli    Xizang  Tangwei 07/12/94  56  78  80  45  259  65
2721030227  Wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
```

这里匹配以 28 开头的所有文本都被删除了，如果使用 n 选项将不会有何输出。

（2）在进行删除时，如果有多个要匹配的条件，此时可以配合使用 e 选项添加多个编辑指令。例如删除所有匹配 lixia 和 Shanxi 的行：

```
#使用 e 选项删除所有匹配 lixia 和 Shanxi 的行
# sed -ne '/[Ll]ixia/d' -e '/Shanxi/d' -e p students
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2721010409  Liwei   Sichuan tangwei 11/21/92  98  88  85  85  356  89
2921050313  Heli    Xizang  Tangwei 07/12/94  56  78  80  45  259  65
2721030227  Wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
```

执行上面这条命令时，sed 将会按选项 e 指定的编辑指令顺序依次执行。

（3）也可以像显示行那样，使用行号范围删除行。例如需要删除前 2 行：

```
# sed '1,3d' students
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2721010409  Liwei   Sichuan tangwei 11/21/92  98  88  85  85  356  89
2921050313  Heli    Xizang  Tangwei 07/12/94  56  78  80  45  259  65
2721030227  Wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
```

注意：在使用形如“/pattern/,n”的范围匹配模式时，应当注意模式匹配到的行和指定的行号之间的大小关系。如果出现大小倒置的情况，sed 只能完成第 1 个匹配行的编辑任务。

5.3.3　插入和修改文本

通过前面几个小节的介绍可以看出，sed 编辑器其实是通过事先设定好的编辑指令对文本进行编辑，整个编辑过程中无须与人进行交互。因此通常将这种编辑过程中无法与人进行交互的编辑器称为非交互式编辑器。作为一个合格的编辑器，插入和修改文本的功能是不可或缺的，本小节将简单介绍如何插入和修改文本。

在指定的行进行插入时，可以使用编辑指令 a 和 i，而替换行则使用编辑指令 c。如果

插入和替换的内容不止一行，可以在非最后一行使用反斜线“\”符号表示插入的内容并未结束。格式如下：

```
[指定的行或模式][aic\]
The first line.\
The second line.\
……
The last line.
```

在上面的格式中，要插入或替换的文本可以位于一行，但是必须用\n 来指定换行的位置。编辑指令后的反斜线指定了文本起始位置。除此之外，为了避免 sed 引起误解，通常将编辑指令和文本放入单引号中。

（1）例如想要在第 1 行后面插入新的一行：

```
#使用编辑指令 a 在第 1 行后插入一个新行
# sed '1a\This is a new line.' students
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
This is a new line.
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  280  70
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
……
```

注意：执行插入和修改行操作时，如果不使用行号和匹配模式指定要编辑的行，sed 会编辑文本的每一行。

（2）也可以使用行首和行尾匹配来定位要插入新文本的位置。例如要在最后一行插入一个新行：

```
#使用行尾匹配并插入新行
# sed '$a\This is a new line.' students
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  280  70
……
2721030227  Wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
This is a new line.
```

（3）如果插入的多行文本位于同一行，需要在每一个需要断行的位置加入换行符“\n”：

```
#使用换行符“\n”插入多行
# sed '3a\The first line.\nThe second line.' students
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  280  70
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
The first line.
The second line.
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
……
```

（4）从上面几个示例中可以看出，插入多行时命令可读性很差，因此通常将编辑指令和文本放入一个脚本文件内，通过 f 选项调用 sed 进行编辑。

例如使用编辑指令 i 在匹配 Hetao 的行前面插入多行内容：

```
#使用 cat 命令查看脚本文件 insert.sed 的内容
# cat insert.sed
/Hetao/i\
The first line.\
The second line.\
The last line.
#使用选项 f 调用脚本文件 insert.sed 执行编辑
# sed -f insert.sed students
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  280  70
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
The first line.
The second line.
The last line.
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
……
```

（5）如果要插入的是一个很长的文本，可以使用 r 指令将要插入的文本读取后再执行插入操作。例如要将文件 ex 中的文本插入到第 3 行之后：

```
#使用 cat 命令查看 ex 的内容
# cat ex
The first line.
The second line.
The last line.
#使用 r 指令读取 ex 的内容并执行插入操作
# sed '3r ex' students
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  280  70
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
The first line.
The second line.
The last line.
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
……
```

由于使用 r 指令是直接将一个文件中的文本插入另一个文本，因此不需要在每一行的结尾加上反斜线。

（6）如果要使用 sed 替换整行，通常使用编辑指令 c。下面是一个使用编辑指令 c 修改行的示例，首先匹配所有以 27 开头的行，然后使用 c 指令将所有匹配到的行替换成新的内容，最后将输出结果重定向到文件“file1”中：

```
#使用 cat 命令查看脚本 modify.sed 的内容
# cat modify.sed
/^27/c\
This line was modified.
#使用选项 f 调用脚本文件执行替换，并将结果重定向到文件 file1 中
# sed -f modify.sed students >file1
#查看文件 file1 的内容
```

```
# cat file1
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  280  70
This line was modified.
……
```

🔔**提示**：在定位的位置插入新行时，只能使用行号和匹配模式指定要编辑的行，不能指定一个范围。

5.3.4　替换文本和其他编辑指令

sed 提供了两个用于替换文本的编辑指令 s 和 y，它们的使用方法相似但功能却不同。它们的格式如下：

```
s/pattern1/pattern2/flag
y/pattern1/pattern2/
```

s 指令将匹配到的 pattern1 用 pattern2 替换，y 指令的功能与其很相似，不同的是 y 指令要求匹配的模式与用来替换的文本的字符数长度必须相等。

开始这个小节之前，我们引入第 2 个示例文件 students2。这个文件中记录了学生 4 门实践课（选修）的成绩，由于添加成绩的每个人都使用不同的分隔方式，因此这个文件看起来很混乱。查看 students2 的内容：

```
#查看示例文件 students2 的内容
# cat students2
2821020225##Liulu$$0::A**B&&0\\
2821020116##Yangrui$$B::C**0&&0\\
2821010321##Xuli$$0::0**B&&D\\
2921020632##Xiayu$$0::0**0&&0\\
2721010409##liwei$$C::0**0&&0\\
2121050313##Heli$$A::0**A&&B\\
2821030215##liyan$$B::B**C&&0\\
```

本小节将以这个文件为例进行操作，讲解替换文本命令的使用方法。

（1）执行替换操作。

首先需要整理这个文件，使用制表符 Tab 替换所有学号和姓名中间的井号“#”。要将所有的“#”替换，s 编辑指令的 flag 选项应该使用 g：

```
#使用指令 s 的 flag 选项 g 执行全文替换
#使用反斜杠屏蔽井号的特殊意义
#命令中的空白为 Tab 制表符
# sed 's/\#\#/  /g' students2 | tee students2.1
2821020225  Liulu$$0::A**B&&0\\
2821020116  Yangrui$$B::C**0&&0\\
2821010321  Xuli$$0::0**B&&D\\
2921020632  Xiayu$$0::0**0&&0\\
2721010409  liwei$$C::0**0&&0\\
2121050313  Heli$$A::0**A&&B\\
2821030215  liyan$$B::B**C&&0\\
```

上面这个示例中，sed 将所有的“#”替换成制表符 Tab 后，将输出交给 tee 命令，tee 命令将 sed 的结果输出到屏幕的同时，也将结果保存在文件 stdents2.1 中。

提示：在命令提示符中输入制表符 Tab 的方法是先按住 Ctrl 键，再按 v 键，然后松开 v 键，再按 Tab 键松开 Ctrl 键，最后松开 Tab 键。

（2）然后继续使用 Tab 将其他的字符替换掉：

```
#使用 cat 命令查看脚本文件 replace.sed 的内容
# cat replace.sed
s/\$\$/        /g
s/\:\:/ /g
s/\*\*/ /g
s/\&\&/ /g
s/\\*$//g
#使用选项 f 调用脚本文件执行编辑
# sed -f replace.sed students2.1 | tee students2.1
2821020225  Liulu       0    A    B    0
2821020116  Yangrui B   C    0    0
2821010321  Xuli        0    0    B    D
……
```

在上面这个示例中，为了保证对齐，在替换姓名后面的“$”符号时，使用了两个 Tab。脚本最后一行，使用了“s/*$//g”，其中“*$”表示在行尾匹配一个或多个“\”，没有指定用任何字符进行替换，就表示需要删除行尾的“\”符号。

（3）仔细检查生成的文件 students2.2 的内容，发现姓名为 liwei 和 liyan 的学生姓名首字母没有大写，而姓名为 Heli 和 Xuli 的学生学号前两位出现了错误。首先将姓名为 liwei 和 liyan 的学生首字母改为大写：

```
#使用 s 指令替换错误的学生姓名首字母
# sed 's/  l/      L/g' students2.2 | tee students2.3
2821020225  Liulu       0    A    B    0
……
2921020632  Xiayu       0    0    0    0
2721010409  Liwei       C    0    0    0
2121050313  Heli        A    0    A    B
2821030215  Liyan       B    B    C    0
```

上面的示例中，匹配要替换的 l 时，将前面的制表符 Tab 也一并匹配，更快地完成了替换。

（4）当然也可以使用词首匹配的办法来替换：

```
#使用词首匹配时，无需将制表符一起匹配
# sed 'y/\<l/\<L/' students2.2
```

（5）还要将学号出现错误的内容改写过来，使用 s 指令的 n 选项修改姓名为 Xuli 的学生学号：

```
#使用 s 指令的 n 选项修改学生的学号
# sed '3s/28/27/1' students2.3
2821020225  Liulu   0    A    B    0
```

```
2821020116  Yangrui B   C   0   0
2721010321  Xuli    0   0   B   D
……
```

上面这个示例中的“3s/28/27/1”表示仅对第 3 行第 1 个匹配的 28 进行替换。使用同样的方法修改 Heli 的学号并保存为 student2.4。

（6）有时需要查看一些肉眼无法看见的控制字符，此时可以使用编辑指令 l 以肉眼可见的方式输出控制字符。例如：

```
#使用选项 l 显示文本中的控制字符
# sed -n '1,$l' students2.4
2821020225\tLiulu\t\t0\tA\tB\t0$
2821020116\tYangrui\t\tB\tC\t0\t0$
2721010321\tXuli\t\t0\t0\tB\tD$
……
```

（7）有时可能需要对指定行或模式匹配之外的行进行操作，此时可以使用“!”指定需要排除的行。

例如显示除了行号为 1 之外的行：

```
#使用!排除行
# sed -n '1!p' students2.4
2821020116  Yangrui B   C   0   0
2721010321  Xuli    0   0   B   D
2921020632  Xiayu   0   0   0   0
……
```

（8）有时可能需要在一行执行多次编辑，此时将多个编辑指令放入大括号“{}”内。例如将实践课成绩中含有 A 或 B 的先输出，其他的后输出：

```
#下面的示例对所有以 2 开头的行执行简单的排序操作
#注意输出内容中有一个空行
# sed -n '/^2/{/[AB]/p; /[AB]/!H; $g; $p}' students2.4
2821020225  Liulu       0   A   B   0
2821020116  Yangrui     B   C   0   0
2721010321  Xuli        0   0   B   D
2921050313  Heli        A   0   A   B
2821030215  Liyan       B   B   C   0

2921020632  Xiayu       0   0   0   0
2721010409  Liwei       C   0   0   0
```

上面这个示例很长，但看起来却很简单，首先匹配以 2 开头的行（即匹配所有行），如果匹配，则执行“{}”内的语句块（大括号内用分号“;”将各独立的语句行分开）。当一条匹配行被送进大括号时，首先匹配这一行是否含有 A 或 B，如果有则打印，反之则将这一行追加到保留空间内。后面接着匹配这是否是最后一行，如果是则用保留空间中的文本替换模式空间中的文本，如果不是最后一行则匹配下一条。最后的指令“$p”则用来打印被前面的指令“$g”替换掉的模式空间的文本。

由于保留空间的第 1 行为空，且使用的是 H 指令（追加到保留空间），因此在输出中的第 6 行为空行。

将上面这条命令拆开等同于：

```
# sed -ne '/[AB]/p' -e '/[AB]/!H' -e '3h' -e '$g' -e '$p' students2.4
```

（9）最后将 students2.4 文件中的 A、B、C、D 转换为数字 85、75、60、50 并保存为 students2.9：

```
#使用 s 指令将文件中的 ABCD 转换为成绩
# sed -e 's/A/85/g' -e 's/B/75/g' -e 's/C/60/g' -e 's/D/50/g' students2.4
>students2.9
```

（10）使用 s 指令替换文本时，可以使用符号&（与单词 and 读音相同）表示匹配的文本本身。例如使用 s 指令匹配并插入内容：

```
#使用 cat 命令查看文件 bj 的内容
# cat bj
Welcome to beijin!
#设置匹配字符串变量 C2
# C2=Welcome
#使用 s 指令在匹配到的字符串前加上 Hello 和一个空格
# sed -n "s/$C2/Hello! &/p" bj
Hello! Welcome to Beijin!
```

注意：由于单引号会忽略变量引用符的引用功能，因此如果在 sed 编辑指令中使用了变量，就不能将编辑指令放入单引号内。

（11）相对于 s 指令而言，替换指令 y 使用相对较少。此处给出一个修改 Heli 的学号时，使用 read 接收用户输入并进行替换的例子：

```
#使用 read 命令接收用户输入，并使用指令 y 替换错误的学号
# read $C; sed "y/21210503/$C/" students2.3 | tee students2.4
29210503
2821020225  Liulu    0    A    B    0
2821020116  Yangrui B    C    0    0
……
2921050313  Heli     A    0    A    B
……
```

（12）编辑指令 y 的另一个重要用途是用来转换大小写，例如：

```
#使用 cat 命令查看文件 bi 的内容
# cat bj
Welcome to Beijin!
#使用 y 指令转换大小写时，一定要将字母对应关系输入正确
# sed 'y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/' bj
WELCOME TO BEIJIN!
```

本小节详细讲解了如何使用 sed 编辑器的替换和修改功能，通过这些示例不难看出 sed 虽然是非交互式编辑器，但使用方法却非常灵活多样。

5.3.5　处理文本中的控制字符

许多人比较喜欢在 Windows 系统中使用文本编辑器编写脚本（例如 UltraEdit 编辑器

等），然后将编写好的脚本使用共享、SFTP 等方式传递到 Linux 系统上执行。这时可能这些文本中存在一些控制字符，这些控制字符可能会影响脚本的运行或文档的使用（Linux 系统不能识别 Windows 中的部分字符）。因此在使用这些文本之前应该删除这些控制字符。本小节将简单介绍如何使用 sed 删除文本中的控制字符。

要删除文本中的控制字符，首先应该查看文本中的控制字符。通常可以使用 cat 命令的 v 选项查看这些肉眼不可见的控制字符：

```
#使用 cat 命令的 v 选项查看 hello.sh 中的控制字符
# cat -v hello.sh
#!/bin/bash^M
#this is a test script^M
^M
echo "Hello"^M
```

在上面这个示例中，“^M”就是一个控制字符，删除这些控制字符的思路是使用 s 将其删除，但在删除之前必须要能产生一个相同的控制字符。

此处以产生“^M”控制字符为例，在终端下先按住 Ctrl 键，然后再按下 V 键，此时终端上将不会有任何字符显示。然后再松开 V 键，并按下 M 键，此时可以看到屏幕上已经产生了控制字符^M，最后松开 Ctrl 键和 M 键即可完成输入。

使用如下命令删除文本中的控制字符：

```
#使用 s 指令将控制字符^M 删除并保存到文件 hello1.sh 中
# sed 's/^M//g' hello.sh >hello1.sh
#使用 cat 命令的 v 选项验证命令的运行结果
# cat -v hello1.sh
#!/bin/bash
#this is a test script

echo "Hello"
```

示例命令中的“^M”一定要手动输入，不能采用复制粘贴的方法。

5.3.6　分支结构

在使用 sed 对文件进行编辑时，可能会遇到在一行进行多次重复修改或按某个特定顺序进行排序的情况。这时情况将变得复杂，通常情况下解决办法是不停地使用大括号“{}”，将不同匹配的语句块进行分割，然而这样会给阅读这条命令的人带来巨大麻烦。

这时建议使用分支结构，对不同功能的语句块进行分割。就像 C 语言中的函数一样，整个过程都十分清晰，并且对某些行还可以跳过不必要的匹配操作，节省了处理时间（这在对大文件进行编辑时十分有用）。本小节将介绍 sed 编辑文本的流程和分支结构的使用方法。

说明： sed 命令的分支结构是一个不常用的功能，因此大多数读者对此部分内容了解即可，不需要的读者可以跳过本小节。

1．sed编辑文本的流程

在使用分支结构之前，先要了解 sed 编辑文本的流程，此处使用上一小节中的按实践课成绩简单排序的例子进行分析：

```
# sed -ne '/[AB]/p' -e '/[AB]/!H' -e '3h' -e '$g' -e '$p' students2.4
```

在这个例子中，sed 首先从文件 students2.4 中读取第 1 行，然后按顺序进行匹配。匹配第 1 条编辑指令，如果匹配，就执行第 1 条编辑指令附带的编辑操作，不匹配就进入第 2 条编辑指令。在第 2 条编辑指令处，同样使用匹配执行的办法编辑模式空间中的内容。然后第 3 条……就这样一直到把所有的编辑指令都匹配执行完成，然后 sed 再读取下一行重复上面的动作，直到文件结束。

从上面这个例子的分析中可以看出，sed 做了很多不需要进行的匹配操作。例如只要第 1 条编辑指令被匹配，就不需要再对第 2 条指令进行匹配了，因为这两条指令的匹配模式是互相排斥的。而只要当前模式空间中的文本不是最后一行，就不需要多次执行最后的匹配操作，由此可见这条命令执行了许多不必要的工作。

分支结构的最重要功能是编辑指令分段，例如只要执行了第 1 条指令就可以读取下一行，不必再执行后面的匹配和编辑指令（类似于流程控制中的中断循环）。另一个例子：只要匹配到文本的最后一行，就可以同时执行指令 g 和 p，不必再多一次匹配（类似于函数的功能）。

2．使用分支结构编辑文本

此处使用 students 文件作为示例，要求 sed 完成以下功能。

- ❑ 对这个文件按入学时间从小到大排序。
- ❑ 将班级 27210103 和 27210104 合并为 27210102。
- ❑ 合并之后在原班级 27210104 的学生学号前加上标记符号“>>”，同时将辅导员姓名改为 Luolei。

将这个过程编写成一个脚本文件，然后使用 f 选项调用：

```
#使用 cat 命令查看脚本文件 sort.sed 的内容
# cat sort.sed
#n                  #禁用默认输出
:a                  #开始标记
s/27210103/27210102/g
t b                 #如果上面的替换语句 s 执行成功，就跳转到 b 处执行
/27210104/b c       #如果匹配到学号 27210104，则跳转到 c 处执行
:b                  #排序开始标记
/^27/p          #如果行首是 27，则直接输出
/^28/{
G; h}           #如果行首是 28，就将保留空间的文本追加到模式空间之后，再用模式空间中的文
本
                #替换保留空间中的内容
/^29/H          #如果行首是 29，就将模式空间中的文本追加到保留空间
$b d            #如果是最后一行，则跳转到 d
n;b a           #读取下一行替换模式空间的文本，跳转到开始标记 a
```

```
:c          #处理 27210104 班学生信息开始处
s/27210104/>>>27210102/g    #修改学号
s/[tT]angwei/Luolei/g
p           #修改辅导员并输出
$b d        #如果是最后一行，则跳转到 d
n; b a      #读取下一行替换模式空间的文本，跳转到开始标记 a
:d
$!b a       #如果不是最后一行，则跳转到开始标记 a
g           #用保留空间中的文本替换模式空间的文本
s/\n\n/\n/g #删除多余的换行符
p;q         #输出并退出
#使用选项 f 调用脚本 sort.sed 执行编辑操作
#sed -nf sort.sed students
2721010221  Xuli    Jiangsu Luolei  12/25/92    76  81  85  79  321 80
>>>2721010209   Liwei   Sichuan Luolei  09/21/92    98  88  85  85  356 89
2721030227  wangtao Yunnan  Huli    03/21/93    87  76  69  88  320 80
2821020115  Liumin  Henan   lixia   05/14/94    78  65  59  78  227 70
2821020225  Liulu   Sichuan Lixia   01/23/93    89  76  88  72  325 81
2921020632  Xiayu   Shanxi  Hetao   03/26/93    78  86  92  78  334 84
2921050313  Heli    Xizang  Tangwei 11/12/94    56  78  80  45  259 65
```

在上面这个示例中，使用了 a、b、c、d 表示 4 个标记点。用这 4 个标记点将整个脚本分割成 4 个不同的功能块。

- 从 a 到 b 标记点为第 1 个功能块：这个功能块主要用来编辑合并的班级。将班级为 27210103 的学生学号修改为 27210102 并跳转到第 2 个功能块排序。将班级 27210104 交由从 c 到 d 标记点之间的第 3 个功能块编辑。
- 从 b 到 c 标记点为第 2 个功能块：主要功能是按学号进行排序。将以 27 开头的学生信息直接输出。如果是以 28 开头，则将保留空间追加到模式空间，最后再替换保留空间，将以 28 开头的学生信息放在保留空间的第 1 行。以 29 开头的学生信息则直接追加到保留空间最后。这样以 27 开头的学生信息被直接输出了，而以 28 和 29 开头的学生信息则按顺序放在保留空间内。
- c 到 d 标记点为第 3 功能块：主要用来对 27210104 班的学生信息进行编辑，修改学号和辅导员并输出。
- 从 d 标记点到最后为第 4 个功能块：这个功能块的主要功能是将保留空间中的文本进行简单编辑然后输出，最后使用 q 指令退出 sed。

在脚本的第 1 行用了“#n”禁止默认输出功能，如果不使用“#n”，则换行时应该使用 D 指令。

这个例子简单地将 b、t 分支语句结合起来，实现对文本的编辑。同时还使用了 G、H 等对保留空间进行和模式空间进行操作的指令。感兴趣的读者可以结合这个例子，对没有涉及的 D、N 和 x 等指令进行练习。

注意： 默认情况下，sed 只有执行完所有编辑指令后才读取下一行，因此使用 b、t 分支语句时，如果要提前结束当前行的编辑，应该使用 n 和 N 等指令读取下一行。

5.4 格式化文本数据抽取工具 awk

在管理和维护 Linux 系统的过程中，有时可能需要从一个具有一定格式的文本（格式化文本）中抽取数据，这时可以使用 awk 编辑器来完成这项任务。发明这个工具的作者是 Aho、Weinberg 和 Kernighan，awk 因此而得名。

与 sed 相比，awk 更擅长处理格式化文本。格式化文本一般使用某个特定的字符（称为域分隔符）将文本中不同的字段（称为域）隔开。例如用于保存用户信息的系统用户文件/etc/passwd，该文件使用冒号分别将用户名、密码、UID 等字段分隔开。本节将介绍如何使用 awk 工具在格式化文本中抽取数据。

提示：对于初学者而言，awk 工具比较难掌握。其原因在于当输入一条错误的 awk 命令时，返回的错误信息让人感到茫然。建议初学者从小的命令开始，逐渐熟悉其用法，从而很快掌握这个命令。

5.4.1 awk 命令基本格式

【命令格式】

```
awk [-F] 'command' input-file
awk -f script input-file
```

与 sed 类似，awk 也有两种调用方式：第一种是直接使用 awk 命令调用，选项 F 用于指定域分隔符。默认情况下 awk 使用的域分隔符为空格，如果要处理的文件 input-file 域分隔符不是空格，应该使用 F 选项另行指定。第二种方法与 sed 一样，先将要输入的选项、模式和动作放入一个脚本文件中，然后使用选项 f 调用。

【命令处理过程】

awk 被调用后，首先读入第 1 行文本并按选项 F 指定的域分隔符将各个字段划开。以/etc/passwd 其中一行为例：

```
user1:x:501:501::/home/user1:/bin/bash
```

处理这个文件时，应该使用选项 F 指定域分隔符为冒号“:”，划分完成后将这一行称为一条记录。一条记录中的各个字段按顺序称为域 1，域 2，域 3……为方便对这些字段进行处理，使用标识符“$1”表示第 1 个字段，“$2”表示第 2 个字段……依此类推。如果要表示整条记录，应该使用标识符“$0”。

划分记录后，awk 会按预定的模式和动作处理记录。处理完一条记录后，又会读取文本的第 2 行重复上面的过程。

【模式和动作】

一个完整的 awk 命令由编辑语句和格式化文本组成。编辑语句由一个或多个模式和动作组成，格式化文本即用户需要处理的文本，可以来自文件，也可以来自命令输出。

与 sed 命令一样，模式用来指定动作执行的文本位置。在 awk 中，模式可以是条件语句、模式匹配、正则表达式等。如果没有指定模式，awk 会对所有记录执行编辑语句。

动作一般放在模式后面的大括号“{}”内，一般是 awk 的内置函数，例如输出函数 print 等。

下面是一个输出文件 passwd 中所有的用户名的示例：

```
#使用选项 F 指定使用的分隔符为冒号
#模式为所有行，动作是使用函数 print 输出一个字符串和域 1
# awk -F: '{print "username:" $1}' /etc/passwd
……
username:ww
username:user1
username:user2
```

上面这个示例命令使用选项 F 指定域分隔符为冒号，后面的命令中并没有指定模式，awk 将默认匹配所有行。放在大括号内的动作使用了 awk 内置函数 print，先输出了一个字符串“username:”，然后使用标识符“$1”输出第 1 个字段。

【文本头、尾表达式】

在 awk 中，有两个特殊的表达式 BEGIN 和 END。BEGIN 语句将在编辑语句开始执行之前运行，通常用来输出文本头信息。END 语句则在所有编辑语句完成之后运行，一般用来输出结束信息和统计数据等。

使用上一节的示例文件 students，制作一个输出学生学号和姓名，并输出文本头、尾的例子：

```
#使用 BEGIN 语句打印包括字段头和分隔符在内的文本头信息
#使用 END 语句打印文本的结束信息
# awk 'BEGIN{print "Student ID      name\n------------------------"}{print
$1"\t"$2}END{print "------END------"}' students
Student ID       name
-----------------------------
2821020225      Liulu
2821020115      Liumin
2721010321      Xuli
……
2721030227      wangtao
----END---
```

从上面这个示例中可以看出 BEGIN、END 在整个命令执行过程中的顺序。

使用 awk 命令处理完成后，会将处理结果输出到标准输出。如果需要保存 awk 命令的输出结果，可以使用重定向到文件的方法，也可以使用 tee 命令保存。

注意：使用 awk 命令时，一定要将命令输入完整，否则 awk 会留在命令行等待继续输入，直到输入完成为止。

5.4.2　正则表达、元字符、运算符和关系运算符

上一小节简单介绍了 awk 命令的基本情况，在本小节中仍然使用前几节中的示例文件

students 作详细讲解。

首先查看示例文件 students 的内容：

```
# cat students
2821020225  iulu    Sichuan Lixia   01/23/93  89  76  88  72  325  81
2821020115  iumin   Henan   lixia   05/14/94  78  65  59  78  280  70
2721010321  uli     Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2921020632  iayu    Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2721010409  iwei    Sichuan tangwei 11/21/92  98  88  85  85  356  89
2921050313  eli     Xizang  Tangwei 07/12/94  56  78  80  45  259  65
2721030227  angtao  Yunnan  Huli    03/21/93  87  76  69  88  320  80
```

在 awk 中，除了可以使用正则表达式和元字符外， awk 还内置了算术运算符、操作符。awk 中常见的算术运算符如表 5.4 所示。

表 5.4　常见的算术运算符

运　算　符	用　　途	运　算　符	用　　途
=	赋值，例如x=y表示将y的值赋给x	y--	使用后减1
x+y	对x和y求和	x%y	求模
x–y	对x和y求差	x+=y	将x+y赋值给x
x*y	对x和y求积	x–=y	将x–y赋值给x
x/y	对x和y求商	x*=y	将x*y赋值给x
–y	负y，单目运算符	x/=y	将x/y赋值给x
++y	加1后使用	x%=y	将x%y赋值给x
y++	使用后加1	x^y	x的y次幂
--y	减1后使用	x^=y	将x^y赋值给x
?:	条件（三目运算符）		

除了表 5.4 中列出的算术运算符之外，awk 还包含了一些关系运算符和操作符。常见的关系运算符和操作符如表 5.5 所示。

表 5.5　常见的关系运算符和操作符

关系运算符和操作符	说　　明	关系运算符和操作符	说　　明
>	大于	!=	不等于
>=	大于等于	==	等于
<	小于	~/pattern/	匹配pattern
<=	小于等于	!~/pattern/	不匹配pattern

同时 awk 也支持逻辑运算符：||（或）、&&（与）、!（非），也可以将这些运算符与括号“()”结合起来使用。下面将结合一些示例，讲解正则表达式和运算符的使用方法。

（1）结合使用正则表达式，输出所有 2007 年入学的学生信息：

```
#先使用/27210/匹配所有 2007 年入学的学生，然后使用输出$0 的方法打印所有行
# awk '/27210/{print $0}' students
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2721010409  Liwei   Sichuan tangwei 11/21/92  98  88  85  85  356  89
2721030227  Wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
```

（2）结合正则表达式，输出所有来自 Sichuan 的学生学号、姓名和总分：

```
#先使用/Sichuan/匹配所有来自 Sichuan 的学生，然后打印这些记录的域 1、域 2 和域 10
# awk '/Sichuan/{print $1"\t"$2"\t"$10}' students
2821020225      Liulu   325
2721010409      Liwei   356
```

上面的示例中，使用了转义字符“\t”（表示制表符 Tab）分隔各个输出的域。

（3）可以指定某个域的取值精确匹配。例如要输出姓名为 Liwei 的学生信息：

```
#使用==精确匹配域 2 的值
# awk '$2=="Liwei"{print $0}' students
2721010409  Liwei   Sichuan tangwei 11/21/92  98  88  85  85  356  89
```

（4）精确匹配时，可以使用“!”表示不匹配。例如要输出不是来自 Sichuan 的学生信息：

```
#使用感叹号!排除精确匹配
# awk '$3!="Sichuan"{print $0}' students
2821020115  Liumin  Henan   lixia   05/14/94   78  65  59  78  280  70
2721010321  Xuli    Jiangsu Luolei  12/25/92   76  81  85  79  321  80
2921020632  Xiayu   Shanxi  Hetao   03/26/93   78  86  92  78  334  84
……
```

（5）也可以使用操作符表示匹配，例如要输出所有辅导员不是 Tangwei 的学生的情况：

```
#使用匹配操作符精确匹配
# awk '$4 !~/[Tt]angwei/{print $0}' students
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  280  70
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
……
```

（6）也可以将匹配操作符与正则表达式一起使用。例如输出所有 93 年以后出生的学生信息：

```
#对域 5 使用操作符和正则表达式进行匹配
# awk '$5 ~/^.......[3-9]/{print $0}' students
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  280  70
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
……
```

（7）使用竖线“|”匹配两边模式之一，例如要输出所有来自 Sichuan 和 Yunnan 的学生：

```
#使用竖线和操作组成或匹配模式
# awk '$3 ~/(Sichuan|Yunnan)/{print $0}' students
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
2721010409  Liwei   Sichuan tangwei 11/21/92  98  88  85  85  356  89
2721030227  Wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
```

（8）在使用“$0”标识符输出所有域时，也可以省略后面的输出动作。下面这条命令和上面的命令效果一样：

```
#省略 awk 命令中的动作
# awk '$3 ~/(Sichuan|Yunnan)/' students
```

（9）可以使用算术运算符对域进行计算，这在制作一些统计信息时非常有用。例如要计算所有公共课的平均成绩并输出：

```
#使用 print 函数重新输出域，并计算公共课的平均成绩
# awk '{print $1"\t"$2"\t"$6"\t"$7"\t"$8"\t"$6+$7+$8"\t"($6+$7+$8)/3}'
students
2821020225    Liulu     89    76    88    253    84.3333
2821020115    Liumin    78    65    59    202    67.3333
2721010321    Xuli      76    81    85    242    80.6667
……
```

（10）可以结合使用关系运算符和逻辑运算符，以实现更复杂的运算。例如要输出所有来自 Sichuan 或者平均分大于 320 的学生信息：

```
#同时使用匹配、逻辑运算符实现更复杂的运算
# awk '($3=="Sichuan" && $11>80){print $0}' students
2821020225 Liulu   Sichuan Lixia      01/23/93  89  76  88  72  325  81
2721010409 Liwei   Sichuan tangwei    11/21/92  98  88  85  85  356  89
```

上面这个例子中，如果记录满足第 3 个字段是 Sichuan，并且第 11 个字段大于 80，则执行输出操作。

（11）使用逻辑运算符也可以实现更复杂的匹配操作。例如输出来自 Sichuan 或出生于 92 年的学生：

```
# awk '($3=="Sichuan" || $5 ~/.......2/){print $0}' students
2821020225 Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
2721010321 Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2721010409 Liwei   Sichuan tangwei 11/21/92  98  88  85  85  356  89
```

注意：使用 awk 时，一定要将多个模式和条件放在括号中，将执行编辑的语句放入单引号内，函数和流控制语句放入大括号内，避免产生错误。

5.4.3　在 awk 命令中使用变量

在 awk 命令中也允许使用变量。awk 中的变量有两种：一种是内置变量，这些变量通常用于控制输出和保存“awk”当前工作状态等信息，在引用变量时通常不需要使用美元符号“$”；另一种是用户自定义的变量，这些变量由用户自己定义并使用，通常自定义变量放在 BEGIN 语句中初始化（赋值）。

使用自定义变量时，如果 awk 不能确定自定义变量的类型，通常将其默认为字符串进行处理。

1．内置变量

常用的内置变量有以下几种。

- FILENAME：用于保存输入文件的文件名称。

- NF：用于保存当前正在处理记录的域个数。
- NR：用于保存从文本中读取记录的个数。
- FNR：用于保存当前读取的记录数。当输入的文件有多个时，读取新文件时，awk 会重置这个变量。
- OFS：用于设置输出分隔字段的字符，默认为空格。
- FS：用于设置字段分隔符。
- ORS：用于设置输出记录分隔符，默认为新的一行。
- RS：用于设置记录分隔符，默认为新行。
- OFMT：数字的输出格式。
- ENVIRON：读取环境变量。

awk 还有许多内置变量，但这些变量都不常用，此处不作赘述，感兴趣的读者可以参考相关文档了解。

（1）有时候可能需要对输出的文本重新格式化，这时可以使用内置变量 OFS 和 ORS，设置输出文本的域分隔符和记录分隔符。

下面以/etc/passwd 文件为示例，输出用户名及使用的 Shell：

```
#使用手动设置分隔符的方法输出用户名和 Shell
# awk 'BEGIN{FS=":";print "username\tShell"}{print NF,NR,$1"\t\t"$7}END
{print FILENAME}' /etc/passwd
Username        Shell
7 1 root        /bin/bash
7 2 bin         /sbin/nologin
7 3 daemon      /sbin/nologin
7 4 adm         /sbin/nologin
……
7 36 user2      /bin/bash
/etc/passwd
```

上面这个示例也可以使用 OFS 设置域分隔符，这样后面的 print 函数就可以不使用“\t\t”手动分隔。

（2）有时会遇到一些使用特殊分隔符的文本。例如下面这个示例中，需要重新设置 RS 才能顺利读取文件：

```
#使用 cat 命令查看示例文件 students3 的内容
#下面这个示例文件使用#作为字段分隔符
#使用空行作为记录分隔符
# cat students3
2821020225#Liulu#0#A#B#0\

2821020116#Yangrui#B#C#0#0\

2721010321#Xuli#0#0#B#D\

2921020632#Xiayu#0#0#0#0\

2721010409#Liwei#C#0#0#0\

2921050313#Heli#A#0#A#B\

2821030215#Liyan#B#B#C#0\
#在 BEGIN 表达式中，重新定义变量 FS、RS 的值，以便能读取到 students3 的内容
```

```
# awk 'BEGIN{FS="#";RS="\\n\n"}{print $1,$2}' students3
2821020225 Liulu
2821020116 Yangrui
2721010321 Xuli
……
```

上面这个命令中使用变量 FS 重新设置读取时使用的域分隔符，使用 RS 重新设置记录分隔符，以读取文件 students3。

（3）当输出的字符为一个浮点数时，可能需要指定浮点数的输出格式，此时可以使用内置变量 OFMT。例如：

```
#使用内置变量 OFMT 重新设置浮点数的输出格式为小数点后两位
# awk 'BEGIN{OFMT="%.2f";print 19.23564}'
19.24
```

提示：使用 awk 命令时，如果没有编辑语句，只有 BEGIN 语句，可以不必输入文件名直接运行。

（4）有时可能在处理过程中需要引用环境变量，此时可以使用 ENVIRON 变量。下面是一个使用 ENVIRON 读取环境变量的示例：

```
#使用内置变量 ENVIRON 读取系统的环境变量
# awk 'END{print ENVIRON["LANG"]}' students
en_US.UTF-8
```

上面这个例子中虽然并没有引用任何文件中的内容，但使用了 END 表达式，因此必须使用文件参数，否则这条命令将变得不完整。

2. 自定义变量

在 awk 命令中除了可以使用内置变量之外，还可以使用自定义变量。自定义变量通常用于统计并计算某个数字字段的结果，也可用于引用字符串。

（1）下面是一个统计当前文件夹中所有文件占用空间的示例：

```
#从命令 ls -l 中读取文件的大小，并使用变量统计所有文件占用的空间
# ls -l | awk 'BEGIN{A=0}{A=A+$5}END{print "The size of all files the current
directory is:"A}'
The size of all files the current directory is:51088
```

在上面这个例子中，首先在 BEGIN 表达式中定义了一个变量 A 并初始化为 0。然后与 ls 命令输出的第 5 个字段相加，得到所有文件占用的空间，最后在 END 表达式中将结果输出。

（2）变量也可以在动作语句之后添加。例如：

```
#变量也可以先使用后定义
# awk '($2==NAME1){print $0}' NAME1="Liwei" students
2721010409     Liwei   Sichuan tangwei 09/21/92  98  88  85  85  356  89
```

提示：awk 中引用变量时，通常不需要使用引用符号。但为了便于理解和阅读，应将变量名称大写并加上引用符号。

5.4.4　在 awk 命令中使用流程控制

在 awk 命令中，还内置了一些流程控制语句（简称流控制语句）。使用这些流控制语句，可以完成更多复杂的抽取任务。本小节将简单介绍 awk 中的流控制语句。

1. awk中的流控制语句语法结构

awk 命令中常见的流控制语句有：if、while、do-while 和 for 语句，这些语句的基本语法结构与 C 语言中的语句类似。

（1）if 语句的基本格式：

```
if (条件表达式)
{
   语句块 1
}
else
{
   语句块 2
}
```

当条件表达式的值为真时（数值 0），执行语句块 1，否则执行语句块 2。如果一个 if 语句不能满足需求，awk 命令也允许将多个 if 语句嵌套使用。

（2）循环是流控制语句中不可缺少的部分，在 awk 中可以使用的循环语句有 while、do-while 和 for。awk 中的 while 语句语法结构如下：

```
while （条件表达式）
{
   语句块
}
```

while 语句执行时，会判断条件表达式的值是否为真。如果条件表达式的值为假，就跳过 while 语句。如果为真，就执行语句块并继续判断条件表达式，直到条件表达式的值为假，才执行 while 之后的语句。

（3）awk 中 do-while 语句语法如下：

```
do
{
   语句块
} while (条件表达式)
```

do-while 语句在执行时，先执行语句块，然后判断条件表达式是否为真。如果表达式为假，则停止执行循环，如果为真，则继续执行语句块，直到条件表达式为假。

循环语句 while、do-while 的区别在于，while 语句只有当条件表达式为真时才执行。而 do-while 语句先执行语句块，然后再判断条件表达式是否为真，即 do-while 语句中的语句块最少要执行一次。

（4）在 awk 中除了上述两个循环语句外，还提供了 for 循环语句，其基本格式如下：

```
for(初始表达式;条件表达式;步长)
{
    语句块
}
```

for 语句执行时，首先执行初始表达式，然后判断条件表达式。如果条件为真则执行语句块，执行完语句块后再执行步长。然后又判断条件表达式，直到条件表达式为假时跳出循环。

（5）除了以上介绍的流控制语句以外，还有 4 个语句用于控制循环的执行过程。

- continue：用于在循环语句中控制循环走向。执行到此语句时，立即从当前执行位置跳转到循环开始处开始下一次循环。此语句用于标识后面的语句已经没有必要执行了，必须立即开始下一次循环执行过程。
- break：用于跳出循环，当执行到此语句时，循环立即停止，然后执行循环后面的语句。此语句用于标识当前这个循环语句已经没有必要再继续执行了，必须马上中止循环过程，执行循环之后的语句。
- next：使 awk 读取文本的下一行。该语句将告诉 awk，当前行已经没有任何价值，需要立即读取文本的下一行。
- exit：如果这条语句没有出现在 END 表达式中，则立即跳出执行 END 中的语句（即直接跳过文件最后一行，执行 END 表达式中的语句）。如果出现在 END 表达式中，则 awk 命令立即停止执行并退出。

提示： 由于循环语句会导致命令变得很长，且影响阅读，因此通常建议将循环语句放入脚本文件中执行。

2．流控制语句用法示例

awk 工具集成的流控制语句功能十分强大，可以使用这些语句完成许多高级操作。下面将通过一些示例简单讲解 awk 中流控制语句的使用方法。

（1）前面学习了正则表达式的匹配，现在可以用 if 语句来执行。例如要输出文件 students 中，所有总成绩 320 以上的学生信息：

```
#使用 if 语句判断域 10 是否大于等于 320，如果是则输出
# awk 'BEGIN{OFS="\t"}{if($10>=320)print $1,$2,$10}' students
2821020225      Liulu   325
2721010321      Xuli    321
2921020632      Xiayu   334
……
```

（2）输出来自 Sichuan 的学生中，总成绩大于等于 320 的学生信息：

```
#使用逻辑运算符，精确匹配的同时判断总成绩是否大于等于 320
# awk 'BEGIN{OFS="\t"}{if( $3=="Sichuan" && $10>=320)print $1,$2,$10}'
students
2821020225      Liulu   325
2721010409      Liwei   356
```

（3）计算 ls 命令传送过来的信息中，所有普通文件的大小和文件夹的个数：

```
#使用 cat 命令查看脚本文件 total.awk 的内容
# cat total.awk
BEGIN{
    #在 BEGIN 表达式中初始化变量 A 和 count
    A=0;
    count=0;
}
{
    #如果域 1 的第 1 个字符为减号-，则将域 5 的值与变量 A 相加
    if($1 ~/^-/)
    A+=$5;
    #如果域 1 的第 1 个字符为字母 d，则变量 count 自加
    if($1 ~/^d/)
    count++;
}
#在 END 表达式中输出计算结果和文件夹数量
END{
    print "total:"A;
    #变量 count-2 是因为 ls 命令输出中含有相对路径的当前目录和上级目录
    print count-2," directories.";
}
#使用 awk 的选项 f 调用脚本计算结果
# ls -al /etc | awk -f total.awk
total:1675924
87  directories.
```

（4）Linux 系统管理员经常需要关注网络延时大小，以免对应用服务的正常运作造成影响。最好的解决办法是每隔一段时间，使用 ping 命令测试与某个服务器的连接，并统计延时。

下面是一个使用 ping 命令发送 4 个 ICMP 数据包测试双向连通性，并用 awk 从中计算最大延时、最小延时和平均延时的示例：

```
#使用 cat 命令查看脚本文件 ping.awk 的内容
# cat ping.awk
BEGIN{
      #使用重新定义变量 FS 的方法设置域分隔符为[、:和]
      FS="[:= ]";
      #初始化变量 AVG、MAX 和 MIN
      AVG=0;
      MAX=0;
      MIN=0;
}
{
   #设置 for 语句执行的次数
   for(I=1;I<9;I++)
   {
      #如果当前为第二条记录，则为变量 MAX、MIN、IP ADDR 赋值
      if(NR==2)
        {
          MAX=$11;
```

```
            MIN=$11;
            IP ADDR=$4;
          }
        #如果当前记录数大于 1 且小于 6
        if(NR>1 && NR<6)
        {
           #计算延时总和
           AVG+=$11;
           #如果当前记录的延时大于变量 MAX 中记录的值，则更新 MAX 中的值
           if($11>MAX)
                MAX=$11;
           #如果当前记录的延时小于变量 MIN 中记录的值，则更新 MIN 中的值
           if($11<MIN)
                MIN=$11;
        }
        #如果记录数大于 6，则跳出循环并执行 END 中的表达式
        if(NR>=6)
           exit;
        #否则读取下一条记录，重新开始循环
        next;
     }
}
END{
    #计算平均延时
    AVG=AVG/4;
    #输出 IP 地址，平均延时，最大延时和最小延时
    print "IP address:",IP ADDR;
    print "Avg:",AVG,"ms";
    print "Max:",MAX,"ms";
    print "Min:",MIN,"ms";
}
#使用 ping 命令发送 4 个测试包，并将输出交给 awk 命令计算
# ping -c4 61.*.*.* | awk -f ping.awk
IP address: 61.*.*.*
Avg: 20.93 ms
Max: 70.2 ms
Min: 4.18 ms
```

为了方便抽取数据，先在 BEGIN 语句中定义了 3 种域分隔符：冒号、等号及空格。由于 ping 命令返回的第 1 行是由 ping 自动生成的简要提示信息，因此 awk 将忽略第 1 行。第 2 个到第 5 个记录中返回了 TTL 值和延时等信息，后面则是一些统计信息，因此 awk 处理过程中只处理第 2 个到第 5 个记录。

awk 命令中的其他循环语句与 for 语句类似，此处不再一一举例，读者可以自行验证其用法。

注意：在本节的示例中，为保证安全，IP 地址中的部分用*号代替。

5.4.5　awk 命令中的函数

awk 命令中的函数分为两种：内置函数和用户自定义函数。内置函数多是一些用来处

理字符串和数学运算的函数，用户自定义函数则是由用户自己定义的功能性函数。

1．内置函数

awk 命令中的函数的用法与 C 语言中函数的用法十分相似。常见的内置函数如表 5.6 所示。

表 5.6　常见的内置函数

函　　数	说　　明
sub(regexp, replacement [, target])	当模式regexp第1次出现时，用replacement替换
gsub(regexp, replacement [, target])	将与模式regexp匹配的内容全部用replacement替换
index(string, find)	返回与find匹配的字符串，第1次在string中出现的位置
length([string])	返回字符串string的长度
match(string, regexp [, array])	返回与regexp字符串第1次相匹配的位置。如果不存在则返回0
split(string, array [, fieldsep])	返回字符串数组元素个数
printf(format, expr1 [, expr2, ...])	格式化输出函数
substr(string, start [, length])	从string中返回从start开始，长度为length的子字符串
tolower(string)	将string中的大写字符转换成小写
toupper(string)	将string中的小写字符转换成大写
print	输出函数
rand()	返回一个0到1之间的随机数
system()	将传递来的字符串按Shell命令执行
atan2(x,y)	四象限的反正切函数
cos(x)	余弦函数（*x*为弧度）
sin(x)	正弦函数
log(x)	*x*的自然对数值
exp(x)	E的*x*次幂
int(x)	将*x*转换为整数类型
sqrt(x)	*x*的平方根

上面列出了常见的一些内置函数，awk 的内置函数不止这些，有兴趣的读者可以阅读相关说明文档了解。

本小节将使用前面整理出来的示例文件 students2。查看文件 students2 的内容：

```
#使用 cat 命令查看示例文件 students2 的内容
# cat students2
2821020225      Liulu    0    A    B    0
2821020116      Yangrui  B    C    0    0
2721010321      Xuli     0    0    B    D
2921020632      Xiayu    0    0    0    0
2721010409      Liwei    C    0    0    0
2921050313      Heli     A    0    A    B
2821030215      Liyan    B    B    C    0
```

下面将利用示例文件 students2，讲解 awk 命令中的内置函数的用法。

（1）使用 sub 函数可以查找第 1 次出现的模式并替换。例如用 sub 函数修改匹配模式

的记录：

```
#使用 sub 函数修改记录
# awk '$1 ~/28210202/{print sub(/Liulu/,"Lilu",$0)"\n"$0}' students2
1
2821020225      Lilu            0       A       B       0
```

上面的命令执行完成后，在第 1 行返回了修改的次数，第 2 行输出了修改后的记录。

（2）利用函数 gsub 将所有匹配到的大写字母 A 替换成数字 95，然后返回一共修改的次数：

```
#使用函数 gsub 替换字母 A，并使用连加计算修改次数
# awk 'BEGIN{N=0}{N+=gsub(/A/,"95",$0);print $0}END{print N}' students2
2821020225  Liulu    0   95  B   0
2821020116  Yangrui B   C   0   0
2721010321  Xuli     0   0   B   D
……
2921050313  Heli     95  0   95  B
2821030215  Liyan    B   B   C   0
3
```

由于函数 gsub 执行一个记录的替换就会返回一次，因此此处使用 N+=gsub，否则只会返回最后一条记录替换的次数。

（3）利用 index 函数可以返回第 1 次匹配字符串的位置：

```
#使用 index 函数计算第 1 个 L 出现的位置
# awk '{print index($0,"L");if(NR==2)exit}' students2
12
0
```

（4）利用 length 函数查看字符串长度：

```
#使用 length 函数计算第 1 个记录的字符串长度
# awk '{print length($0);if(NR==2)exit}' students2
25
27
```

（5）使用 match 函数查看匹配模式首次出现的位置：

```
#使用 match 函数计算匹配模式首次出现的位置
# awk 'BEGIN{print match("Hello! Welcome to Beijin!","B")}'
19
```

（6）利用 split 函数将字符串按指定的分隔符拆开，然后放入指定的数组中并返回数组的下标：

```
#先使用 split 函数将一个序列号按指定的分隔符“-”拆开，并存放到数组 ARR 中
#然后使用 for 语句输出数组
# awk 'BEGIN{print split("FV7H8-42D17-08D0Q-8QZG9-YPUZA",ARR,"-");for(I in
ARR)print ARR[I]}'
5
FV7H8
42D17
```

```
08D0Q
8QZG9
YPUZA
```

用 split 函数将序列号进行分割，将分割之后的数据存入数组 ARR，最后利用 for 语句将数组输出。

提示：在 awk 中使用数组时，可以不用先定义，也不必指出元素的个数。

（7）利用 tolower 函数将字符串中的大写字符转换成小写：

```
#将匹配记录中的大写字符全部转换为小写
# awk '$1=="2821020225"{print tolower($0)}' students2
2821020225      liulu          0       a       b       0
```

（8）利用 toupper 函数将字符串中的小写字符转换成大写：

```
#使用 toupper 函数将匹配记录中的字符全部转换成大写
# awk '$1=="2821020225"{print toupper($0)}' students2
2821020225      LIULU          0       A       B       0
```

（9）利用 rand 函数返回一个 0 到 1 之间的随机数：

```
#使用 rand 函数产生一个 0～1 之间的随机数
# awk 'BEGIN{print rand()}'
0.237788
```

（10）sin、cos 函数可以计算正弦和余弦值：

```
#使用 sin、cos 函数计算正弦和余弦值
# awk 'BEGIN{PI=3.1415926;print sin(PI/4),cos(PI/4)}'
0.707107 0.707107
```

（11）使用 log 函数可以返回自然对数：

```
#使用 log 函数计算自然对数
# awk 'BEGIN{print log(100)}'
4.60517
```

（12）exp 函数可以返回 e 的 x 次幂 e^x：

```
#使用 exp 函数计算 e 的 256 次幂
# awk 'BEGIN{print exp(256)}'
1.51143e+111
```

（13）sqrt 函数可以计算平方根：

```
#使用 sqrt 函数计算 256 的平方根
# awk 'BEGIN{print sqrt(256)}'
16
```

（14）在前面的 awk 命令示例中，使用的输出函数一直是 print，它仅能应用一些简单的输出。在 awk 中还提供了一个格式化输出函数 printf，这个函数与 C 语言里的输出函数 printf 一样，可以使用修饰符对输出的内容进行格式控制，其基本语法如下：

```
printf format, [ARGUMENT]...
```

printf 常用的修饰符如表 5.7 所示。

表 5.7　printf常用的修饰符

修　饰　符	说　　明	修　饰　符	说　　明
%c	作为一个ASCII字符输出	%d	作为一个整数输出
%s	作为一个字符串输出	%e	作为一个科学型浮点数输出
%o	作为一个八进制数输出	%f	作为一个浮点型数字输出
%x	作为一个十六进制数输出	%g	由awk决定浮点数输出的形式

除了以上一些修饰符以外，awk 也允许像 C 语言那样使用“-”、域宽和“.”进一步细化输出的格式。

使用 printf 函数将数字 100 分别转换成字符、八进制数和十六进制数：

```
#将数字 100 强制转换为字符、八进制数、十六进制数并输出
# awk 'BEGIN{A=100;printf "%c\n%o\n%x\n",A,A,A}'
d
144
64
```

（15）使用 printf 的修饰符可以将生成的数据转换成相应的格式：

```
#使用不同的修饰符将数据转换为不同的格式
# awk 'BEGIN{printf "%e\n%f\n%g\n",exp(30),exp(30),exp(30)}'
1.068647e+13
10686474581524.462891
1.06865e+13
```

显然在上面的例子中使用%f 是不合适的。

（16）有时在计算一个数字时要保留一定的精度，这时可以使用“.”指定精度，例如：

```
#使用%f 修饰符指定精度
# awk 'BEGIN{printf "%4.5f\n",sqrt(47)}'
6.85565
```

（17）从/etc/passwd 中抽取用户名和用户使用的 Shell：

```
#使用%s 对齐输出
# awk 'BEGIN{FS=":";printf "%-15s %-15s\n","username","Shell";printf
"------------------------------\n"}{printf "%-15s %-15s\n",$1,$7}'
/etc/passwd
username       Shell
------------------------------
root           /bin/bash
bin            /sbin/nologin
daemon         /sbin/nologin
adm            /sbin/nologin
……
```

在上面的示例中，由于使用了格式化的输出，看起来非常规整。

（18）为了使各输出的段对齐，这里使用修饰符。前面计算公共课成绩的例子，此时可以改为：

```
#使用%s 修饰符修改输出格式
# awk '{printf "%-10s\t%-10s\t%d\t%d\t%d\t%d\t%d\n",$1,$2,$6,$7,$8,$6+
$7+$8,($6+$7+$8)/3}' students
2821020225      Liulu       89      76      88      253     84
2821020115      Liumin      78      65      59      202     67
2721010321      Xuli        76      81      85      242     80
……
```

2．用户自定义函数

在 awk 命令中，还允许使用自定义函数。与 C 语言中的函数作用范围不同，自定义函数无论出现在命令中的什么位置，作用都是全局的，而函数中定义的变量只作用于函数内部。如果需要从函数中返回一个值，可以使用 return 语句。

在 awk 命令中自定义函数的基本格式如下：

```
function name([参数列表])
{
     语句块;
     [return ..];
}
```

在调用函数时，只需要使用函数名并加上要使用的参数列表即可，无须使用特别的语句。

（1）下面是一个计算文件 students 中学生公共课的总成绩和平均成绩的例子：

```
#使用 cat 命令查看脚本文件 ex.awk 的内容
# cat ex.awk
#定义函数 Add 用于返回 3 个数之和
function Add(A,B,C)
{
  return A+B+C;
}
#定义函数 Avg 用于返回 3 个数的平均数
function Avg(A,B,C)
{
  return (A+B+C)/3;
}
#使用修饰符修改输出格式的同时，调用 Add 和 Avg 函数计算公共课的总成绩和平均成绩
{
  printf
"%-10s\t%-10s\t%d\t%d\t%d\t%d\t%d\n",$1,$2,$6,$7,$8,Add($6,$7,$8),Avg($
6,$7,$8);
}
#在 END 表达式中输出文件名
END{
printf "%s\n",FILENAME;
}
#使用 awk 的选项 f 调用脚本文件
# awk -f ex.awk students
2821020225      Liulu       89      76      88      253     84
2821020115      Liumin      78      65      59      202     67
```

```
……
students
```

在脚本文件 ex.awk 中，定义了两个函数 Add 和 Avg，分别用来计算公共课的总成绩和平均成绩。其后的编辑语句中，函数 printf 通过调用这两个函数，返回公共课的总成绩和平均成绩并输出。

（2）将上一节中生成的文件 students2.9 中的学生的成绩进行筛选，只输出学号、学生姓名和实践课的最高成绩并重新保存为 students2。

为实现上述功能，此处可以使用条件三目运算符和数组两种方法。为此引出两个脚本文件 stu.awk1 和 stu.awk2：

```
#使用 cat 命令查看 stu.awk1 的内容
# cat stu.awk1
#定义函数 Max
function Max(Arr)
{
   #将数组中第一个元素值赋值给 Ma
   Ma=Arr[1];
   #使用 for 语句查找数组 Arr 中保存的最大数字
   for(I in Arr)
   {
      #如果当前数组元素的值大于 Ma，则将数组元素的值保存到 Ma 中
      if(Ma<Arr[I])
         Ma=Arr[I];
   }
   #返回找到的最大数字 Ma
   return Ma;
}
{
   #使用 for 语句将 5 个域的数值存入数组 Ar 中
   for(J=1;J<5;J++)
     Ar[J]=$(J+2);
   #在 printf 中调用 Max 函数计算最大值并返回
   printf "%-10s\t%-10s\t%d\n",$1,$2,Max(Ar);
}
#查看脚本文件 stu.awk2 的内容
# cat stu.awk2
#定义函数 Max
function Max(A,B,C,D)
{
   #使用三目运算符，比较 A、B 和 C、D 中的最大值，并分别赋值给 A 和 C
   A>B?A:A=B;
   C>D?C:C=D;
   #使用三目运算符，比较 A、C 中的最大值，并赋值给 Ma
   Ma=(A>C)?A:C;
   #使用 return 返回最大值
   return Ma;
}
{
   #使用 printf 修饰输出的同时，调用函数 Max 计算最大值
   printf "%-10s\t%-10s\t%d\n",$1,$2,Max($3,$4,$5,$6);
```

```
}
#使用 awk 的选项 f 调用 stu.awk 脚本，并将结果保存到 students2 中
# awk -f stu.awk1 students2.9 >students2
2821020225      Liulu       85
2821020116      Yangrui     75
2721010321      Xuli        75
……
```

在脚本文件 stu.awk1 中，先用一个循环将第 3、4、5、6 字段放入数组中。然后调用函数 Max，将数组传递给函数，函数利用循环的方法找出最大值然后返回。

在第 2 个脚本文件 stu.awk2 中，先将 4 个参数直接传递给 Max 函数，函数运用三目运算符求出最大值，最后返回最大值。三目运算符 A>B?A:A=B 的含义是：假如 A>B 成立，则返回 A（即不执行任何操作），否则就执行 A=B（把 B 赋值给 A）。

注意： 在 C 语言中 A>B?A:A=B 可以简写成 A>B?:A=B，但在 awk 中这是不允许的。

（3）在本小节的最后，将引用一个使用 awk 监视磁盘并向用户发出警告的例子。在这个例子中，当某一个文件系统空闲空间不足 10%时，通过 system 函数运行 mail 命令向管理员发出警告信息。

使用的脚本和执行过程如下：

```
#使用 cat 命令查看脚本文件 df.awk 的内容
# cat df. Awk
#定义函数 mess，用于向用户发送警告消息
function mess(A,B,C)
{
   #如果使用率大于等于 0.9，则向用户发送警告信息
   if(B>=0.9)
   {
      #使用 system 函数执行 Shell 命令
      #先使用 date 命令将当前时间、磁盘使用情况放入临时文件 df.tmp 中
      #然后使用 mail 命令将 df.tmp 中的内容发送给 c******@163.com 和 root
      #最后使用 rm 命令删除使用过的临时文件
      system("date +'%F %r'>/root/df.tmp; df -h>>/root/df.tmp; cat /root/
      df.tmp|mail -s 'Disk Warning' c******@163.com,root;rm -rf /root/
      df.tmp");
      #使用 printf 函数输出警告消息
      printf "Disk Warning!\nFile System:%s\nUsed:%3.0f%\nMounted on:",
      A,B*100,C;
   }
}
{
   #判断当前行是否以分区开头
   if($1 ~/^\/dev\/sd/)
   {
      #计算使用率
      N=$3/$2;
      如果使用率大于等于 0.9，则调用函数 mess
      if(N>=0.9)
         mess($1,N,$6);
```

```
    }
}
#将命令 df 的输出交给 awk 处理
# df | awk -f df.awk
#以下为命令输出
Disk Warning!
File System:/dev/sda2
Used:90%
#以下为使用 mail 命令查看邮件的过程
# mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/spool/mail/root": 1 message 1 new
>N  1 root@localhost.local  Sun Aug  8 00:03  21/899   "Disk Warning"
#按下回车键读取第 1 封邮件
&
#以下为邮件内容
Message 1:
From root@localhost.localdomain  Sun Aug  8 00:03:23 2010
Date: Sun, 8 Aug 2010 00:03:16 -0700
From: root <root@localhost.localdomain>
To: c******@163.com, root@localhost.localdomain
Subject: Disk Warning

2010-08-08 12:03:16 AM
Filesystem       Size    Used    Avail   Use%    Mounted on
/dev/sda2        3.8G    2.7G    1.1G    90%     /
/dev/sda3        15G     166M    14G     2%      /home
/dev/sda1        46M     11M     33M     24%     /boot
Tmpfs            252M    0       252M    0%      /dev/shm
```

在上面的示例中，awk 首先匹配所有 SCSI 磁盘设备的文件系统，然后计算匹配到的文件系统使用率。当使用率大于等于 90%时调用 mess 函数，将文件系统、使用率和挂载点 3 个参数传递给 mess 函数。mess 调用 awk 的内置函数 system 将系统当前时间和磁盘整体使用情况放入临时文件/root/df.tmp 中。后面的 mail 命令将临时文件的内容用邮件发送给 c******@163.com 和 root，删除临时文件后，输出警告信息到标准输出。

注意：在调用函数 system 时，一定要将 Shell 命令放入双引号中，Shell 命令原本应放入双引号中的参数此时应该放入单引号中。

5.5　转换和删除重复命令 tr

在本章的前面几节中介绍了几个用于处理字符的命令和工具，然而在处理大小写转换、删除重复字符等任务时，这些命令处理起来相对较为麻烦。在本节中将简单介绍 Linux 下自带的 tr 命令，相对其他命令而言，其语法较为简单，比较方便记忆，即使初学者也可以轻松掌握这个命令。

5.5.1　tr 命令的基本格式

tr 命令用于处理字符转换、删除重复字符等任务，可能这并不是一个常用的命令，但

在许多时候，tr 命令可以高效地完成任务，因此初学者应该了解这个命令的基本用法，以备不时之需。

【命令格式】

```
tr [options] [string1] [string2]
```

在 tr 命令的基本格式中，string1 用于查询字符串，string2 用于转换查询到的字符串。与本章前面介绍的几个工具不同，tr 命令不能读取文件，因此如果要转换文件中的内容，可以使用重定向输入或管道的方法。

【常用选项】

tr 命令常用的选项及其含义如下。

- c：使用 string1 字符集中的补集替换，要求的字符集为 ASCII。
- d：删除 string1 中包含的所有字符。
- s：将所有重复出现的字符进行压缩，只保留一个字符，即删除重复的字符。

在使用 tr 命令时，还需要指定 string1 和 string2 这两个格式化字符串，字符串的形式可以是字符的范围、单个字符、多字符列表等。常见的字符串形式及其含义如下。

- [a-z]：小写字母 a～z 范围内的字符及其组合而成的字符串。
- [A-Z]：大写字母 A～Z 范围内的字符及其组合而成的字符串。
- [0-9]：数字 0～9 范围内的字符及其组合。
- [C*n]：这个表达式中，C 表示字符或字符组合，*n* 表示重复出现的次数。即字符或字符组合出现 *n* 次的字符串。

除了以上这些常见的字符串形式以外，tr 命令还支持在字符串中查询和处理一些特殊的字符，这些字符通常都使用一个八进制数字表示。常见的特殊字符及其对应的八进制表示形式如下。

- \012：新的一行，在其他命令中经常使用“\n”表示。
- \011：相当于一个 Tab 键，有些命令中使用“\t”表示。
- \015：Enter 键（即换行符），在其他命令中经常使用“\r”表示。
- \014：新的一页，在其他命令中经常使用“\f”表示。

除此之外还有一些不常见的特殊字符，由于很少用到，此处不作赘述，感兴趣的读者可以自行参考相关文档。

注意：tr 命令的两个格式化字符串的常见形式，与正则表达式非常类似，仅有少许差别，初学者需要特别注意。

5.5.2　字符转换

使用 tr 命令时，最常见的任务是对一个字符串执行转换。本小节将简单介绍 tr 命令的字符转换功能。

（1）最常见的字符串转换是大小写转换。例如需要将一个字符串中的字符全部转换为大写：

```
#使用 tr 命令的选项 s 将小写字母转换为大写字母
```

```
# echo "this is a test string." | tr -s "[a-z]" "[A-Z]"
THIS IS A TEST STRING.
```

从上面的输出中可以看到，tr 命令已经将所有的小写字母全部转换为大写字母。

（2）由于 tr 命令不能直接从文件中读取并转换文本，因此通常都使用重定向输入或管道传递的方法。例如需要将文本文件 test 中的字符执行大小写转换：

```
#使用 cat 命令查看文件 test 的内容
# cat test
this is the first line.
this is the second line.
#使用重定向输入的方法转换文件 test 中的内容
# tr -s "[a-z]" "[A-Z]" <test
THIS IS THE FIRST LINE.
THIS IS THE SECOND LINE.
```

（3）除了可以使用[a-z]、[A-Z]这样的形式外，还可以使用一个列表的形式。例如需要将文件 test 中的小写字母 tfls 转换成大写字母：

```
#使用一个列表作为查询和转换字符串
#注意此处命令中的列表与正则表达式中的列表的区别
# tr -s "tfls" "TFLS" <test
ThiS iS The FirST Line.
ThiS iS The Second Line.
```

（4）除此之外，tr 命令还可以对文本进行格式转换，例如下面这个文本文件：

```
#查看文件 test1 的内容
#文件使用#号作为分隔符
# cat test1
2821020225#Liulu#0#A#B#0\
2821020116#Yangrui#B#C#0#0\
2721010321#Xuli#0#0#B#D\
2921020632#Xiayu#0#0#0#0\
2721010409#Liwei#C#0#0#0\
2921050313#Heli#A#0#A#B\
2821030215#Liyan#B#B#C#0\
```

要重新转换格式，可以采用将文本中的“#”转换为 Tab 制表符的办法：

```
#使用选项 s 和分隔符转换文件格式
# tr -s "#" "\t" <test1
2821020225      Liulu   0       A       B       0\
2821020116      Yangrui B       C       0       0\
2721010321      Xuli    0       0       B       D\
……
```

5.5.3　删除字符

tr 命令的另一个重要功能是用来删除字符，包括删除重复字符和自定义的字符集等。本小节将简单介绍使用 tr 命令删除字符。

（1）使用 tr 命令删除重复字符时，需要指定字符串或文本中可能存在重复的字符集合，因此仅需要使用 string1 即可。

例如需要删除由于一些误操作输入的重复字母：

```
#使用选项 s 删除指定的重复字母集
# echo "HHHHeeeelllooo" | tr -s "Heo"
Hello
```

（2）可以使用一个巨大的字符集删除可能存在的重复字符。例如要删除所有字母的重复：

```
#使用[a-z][A-Z]表示删除所有字母的重复
#注意此处与正则表达式的区别
# echo "HHHHeeeelllooo" | tr -s "[a-z][A-Z]"
Helo
```

（3）也可以删除文本中的一些特殊字符，例如删除文本文件中多余的空行：

```
#查看示例文件 test2 的内容
# cat test2
a

b


c
#使用\n 表示空行，删除多余的空行
# tr -s "\n" <test2
a
b
c
```

（4）也可以使用八进制数表示特殊字符。删除多余空行的命令也可以写为：

```
#使用八进制数表示换行符
# tr -s "\012" <te1
```

（5）tr 命令还可以用来删除特定的字符集：

```
# echo "My name is Jhon." | tr -d "msn"
My ae i Jho.
```

从上面的结果可以看出，tr 命令在处理替换和删除时，采用了逐字符查询处理的方法。

注意：由于 tr 命令执行时会逐字符查询处理，因此使用时应该特别注意字符集 string1 的选择。

5.6　合并和分割工具

许多时候需要对文件进行排序、合并和分割，例如要将两个有关联的文件内容连接起来。Linux 系统中有几个命令可以实现这些操作，包括 sort、cut、paste、join、uniq 和 split。本节将通过一些示例详细讲解这些工具。

5.6.1　排序命令 sort

UNIX 和 Linux 自带的 sort 命令功能非常强大，其主要功能是对文本内容按不同的方法排序。它不仅可以按一个或多个字段排序，还可以合并文件。使用 sort 处理一些较大的文件时，可能处理速度会比较慢，但却非常有效。本小节将简单介绍如何使用 sort 命令排序、合并文件。

【命令格式】

```
sort [option] [file]
```

sort 处理的文本可以来自一个文本文件，也可以来自标准输入和管道等。

【常用选项】

sort 是一个功能强大而实用的命令，常用的选项如表 5.8 所示。

表 5.8　sort命令的常用选项

选　项	选 项 含 义
b	按字段进行分类并忽略前面的空格或制表符
d	按字典的顺序进行排序，将除空格和字母以外的字符排除
f	排序时将小写字母和大写字母平等对待，即忽略大小写
g	根据数值进行排序
i	只考虑可打印的字符
M	将字符按月份进行比较，例如JAN<...<DEC
n	按数值进行比较排序
r	反向排序
c	测试文件内容是否已经进行了排序，若无任何返回信息，则表示已经执行了排序操作
k	指定排序的关键字
m	合并已经排序的文件，不进行排序
o	将结果写入文件内，不再输出到标准输出上
s	通过屏蔽最后的分类比较稳定排序
t	使用指定的字符作为字段分隔符
T	将临时文件放入指定的目录内
u	如果与选项c一起使用，则检查是否在排序时已经去除重复的行，没有选项c时，则在排序时去掉重复的行
z	用一个0字节作为结束，而不是一个换行符

默认情况下，sort 像 awk 那样用空格或制表符 Tab 来分隔各个字段。如果要排序的文件不是以空格或制表符作为字段分隔符，应该使用 t 选项指定分隔符。分隔后就可以按字段进行排序了。排序时既可以按某个字段排序，也可以按整行排序。

【用法示例】

（1）在使用 sort 命令对文本进行排序时，sort 默认会将排序结果输出到标准输出。如果要保存到文件，可以使用重定向或选项 o。例如：

```
#排序文件并将结果重定向到文件 students_sor 中
```

```
# sort students >students_sor
#查看文件 students_sor 的内容
# cat students_sor
2721010321  Xuli     Jiangsu Luolei   12/25/92  76  81  85  79  321  80
2721010409  Liwei    Sichuan tangwei  09/21/92  98  88  85  85  356  89
2721030227  wangtao  Yunnan  Huli     03/21/93  87  76  69  88  320  80
……
#使用选项 o 将排序结果保存到文件 students_sor 中
# sort -o students_sor students
```

使用 o 选项保存结果时，如果指定的保存位置和输入的文件是同一个文件，sort 会先将结果保存在临时文件内，等排序结束才将结果写入。

提示： 应该尽量避免将输出保存到输入的文件中，特别是对一些较长的文件进行操作时，意外的错误可能会导致数据丢失。

（2）在对文件进行合并、连接之前，应该先检查文件是否已经执行了排序，对于一些小文件，如果人为进行查看不是很困难，但是对于有几百行文本的文件，这样做是非常困难的。这时可以使用 c 选项检测文件是否已经进行了排序。例如：

```
#使用选项 c 检查文件 students_sor 是否已排序
# sort -c students_sor
#无任何输出表示已排序
#
#使用选项 c 检查一个未排序的文件
# sort -c /etc/sysconfig/iptables
sort: /etc/sysconfig/iptables:3: disorder: *filter
```

如果文件已经排序，sort 将不会返回任何结果，若没有排序，则返回相关提示信息。

（3）有时文件的行前面可能会出现一些特别的字符，影响排序的结果。这时查看文件内容并结合使用选项 b 和 d 可以排除这些字符的干扰。例如：

```
#使用选项 b 排序时忽略空格和制表符
# sort -b name2
1caili
%Biangang
       Fengjia
……
#使用选项 d 只排序数字和字母
# sort -d name2
1caili
%Biangang
       Fengjia
……
```

（4）在某些系统中，sort 排序会对大写字母和小写字母区别对待。这时为了避免在手动输入时产生的失误，需要将小写字母和大写字母同等对待，可以使用 f 选项。例如：

```
#使用选项 f 将大小写字母同等对待
# sort -f char
a
```

```
b
c
d
E
F
G
```

（5）默认情况下，sort 总是先查看每 1 行的第 1 个字符，如果第 1 个字符相同，则比较第 2 个字符，依此类推。但有数字出现的情况则不同，默认情况下，数字 1 会排在数字 2 前面。

如果统计数字，可能用户更希望 sort 按数值从小到大排序，此时应该使用选项 g 让“sort”按数值从小到大进行排序。例如：

```
#使用选项 g 按数值排序
# sort -g name3
1
2
67
100
```

（6）有时为了统计一些数据，需要按月份进行排序，比如一个含有销售数据的文本，需要按月份统计销量或金额，此时可以配合使用选项 M：

```
#使用选项 M 将字符串转换为月份进行排序
# sort -M sales
Jan 349 837.6
Feb 463 1111.2
Mar 597 1432.8
Apr 654 1569.6
May 365 876
Jun 483 1159.2
Jul 489 1173.6
……
```

（7）在阅读一个大文件或按某些特定的字段进行排序之后，可能更关注文件或文本的末尾（文本的末尾往往有一些统计信息或数值比较大的数据）。这时可以使用选项 r，让 sort 对文本反向排序。例如：

```
#使用选项 r 反向排序（即倒序）
# sort -rM sales
Dec     621     1490.4
Nov     587     1408.8
Oct     645     1548
Sep     498     1195.2
Aug     325     780
Jul     489     1173.6
Jun     483     1159.2
……
```

（8）在上面的几个例子中，sort 都使用第 1 域作为排序的关键序。然而有时我们关心的内容不在第 1 个字段，这时就要用到选项 k 对指定域排序。

例如要在 students 中按学生辅导员进行排序：

```
#使用选项 k4 指定第 4 域为排序关键域
# sort -k4 students
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2721030227  wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
……
```

（9）有时可能对排序的要求更严格，一个典型的例子是某个字段中一个或多个字符指明了产品的产地。这时可以先用一个数字确定要排序的关键域，然后用点号“.”加域中的第几个字符指定排序关键字符。

例如在学生信息文件 students 中，按学生出生的年份进行排序，出生日期在文本的第 5 个字段，而年份在第 5 个字段的第 8 个字符。因此可以使用以下命令：

```
# sort -k5.8 students
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2721010409  Liwei   Sichuan tangwei 09/21/92  98  88  85  85  356  89
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2721030227  wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
……
```

（10）使用选项 k 对某个特定的字段排序时，如果要对这个字段中的数字按从小到大排序，应该使用选项 n 按数字排序。例如要将文本 students 中的学生信息按总成绩从大到小排序：

```
#使用选项 n 对选项 k 指定的域 10 按数字排序
# sort -k10nr students
2721010409  Liwei   Sichuan tangwei 09/21/92  98  88  85  85  356  89
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
……
```

（11）在使用选项 k 时，也可以指定多个字段排序，方法是使用逗号“,”将多个字段分开，也可以用多个选项 k 指定字段。

例如对文件 students 文件中的学生按出生年份对总成绩进行排序：

```
#下面的命令会先按出生年份排序，若年份相同，再按总成绩排序
# sort -k 5.8,10n students
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2721010409  Liwei   Sichuan tangwei 09/21/92  98  88  85  85  356  89
2721030227  wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
……
```

上面这个示例中，sort 先对年份排序，然后再对成绩排序。下面这条命令与该命令是等价的：

```
#使用两个 k 选项分别指定两个排序关键域
# sort -k 5.8n -k 10n students
```

（12）有时要处理的文本并不像上面的文本那样，以空格或制表符作为字段分隔符，这时应该使用选项 t 指定分隔符排序。例如：

```
#使用选项 t 指定#为字段分隔符，并使用第 3 域作为关键域排序
# sort -t# -k3 students4
2921020632#Xiayu#0#0#0#0
2721010321#Xuli#0#0#B#D
2821020225#Liulu#0#A#B#0
……
```

（13）当文本中有一个或多个重复的行时，可以使用选项 u 检查或去除重复的行。例如下面这个例子：

```
#使用 cat 命令查看 stu_name 的内容
# cat stu_name
Heli Xizang Tangwei
Liulu Sichuan Lixia
Liumin Henan lixia
Liumin Henan lixia
liwei Sichuan tangwei
Liwei Sichuan tangwei
wangtao Yunnan Huli
Xiayu Shanxi Hetao
Xuli Jiangsu Luolei
#使用选项 c 检查文件 stu_name 是否已经排序，命令无任何输出
# sort -c stu_name
#使用选项 cu 检查文件是否排序并存在重复
# sort -cu stu_name
sort: stu_name:4: disorder: Liumin Henan lixia
```

（14）也可以使用选项 u 对文本中的重复行执行删除操作：

```
#使用选项 u 删除文本中的重复行
# sort -u stu_name
Heli Xizang Tangwei
Liulu Sichuan Lixia
Liumin Henan lixia
liwei Sichuan tangwei
……
```

（15）还可以使用 u 和 f 选项在删除重复行的时候忽略大小写：

```
#使用选项 f 删除重复行时，忽略大小写
# sort -uf stu name
Heli Xizang Tangwei
Liulu Sichuan Lixia
Liumin Henan lixia
……
```

（16）sort 命令还可以合并文本，但通常推荐合并的多个文本应该具有相似的结构，以便于合并之后阅读或使用。

例如现在将两个保存有学生信息的文件 students_1.sort 和 students.sort 合并：

```
#使用选项 m 合并文件 students.sort 和 students_1.sort，并保存到 students_new 中
# sort -m students.sort students_1.sort >students_new
#查看合并后的文件内容
# cat students_new
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2721010409  Liwei   Sichuan tangwei 09/21/92  98  88  85  85  356  89
2721030227  wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  227  70
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
……
```

在上面的示例中，sort 先将两个文件合并，然后执行排序并将结果重定向到文件 students_new 中。

🔔注意：在对文件执行合并操作之前，应该先对要合并的文件排序。

（17）管理和维护系统时，为了更直观地查看命令输出信息，也经常用到 sort 命令对其他命令的输出排序。例如查看/etc 目录中最大的 5 个文件：

```
#先使用 ls -l 命令查看目录/etc 中的文件信息并将结果交给 awk 抽取权限、文件大小和名称域
#然后使用 sort 命令对文件大小进行排序并使用 tail 显示前 5 行
# ls -l /etc | awk '{print $1,$5,$9}' | sort -n -k2 | tail -5
-rw-r--r-- 31303 jwhois.conf
-rw-r--r-- 37282 ld.so.cache
-rw-r--r-- 234949 prelink.cache
-rw-r--r-- 362031 services
-rw-r--r-- 807103 termcap
```

🔔注意：sort 命令在不同的系统上运行，产生的结果可能会有差异，此时应该尝试多个选项或者查看相关文档。

5.6.2　数据剪切命令 cut

在 Windows 中，经常从一个文件、可编辑窗体或资源管理器中使用剪切命令，将一段文本移动到其他文本中。在 Linux 中执行这个任务的是 cut 和 paste 命令。本小节将简单介绍数据剪切命令 cut。

【命令格式】

```
cut [option] [file]
```

【常用选项】

- ❑ b：表示要操作的对象是字节。
- ❑ c：表示要操作的对象是字符。
- ❑ f：表示要操作的对象是字段。
- ❑ d：指定字段分隔符，默认情况下是制表符 Tab。
- ❑ s：表示不包含没有字段分隔符的行。这个选项通常用于去掉注释等。

剪切的范围表示方法如下。

- N：表示第 N 个字节、字符或字段。
- N-：表示从 N 到一行结束内的所有文本。
- N-M：表示从 N 到 M 之间的所有文本。
- -M：表示从开始到 M 之间的所有文本。
- -：从开始到结束的所有文本。

除了上面表示范围的方法以外，还可以像前面匹配行时一样，使用诸如“N,M”之类的表示方法。

【用法示例】

此处使用上一小节中生成的示例文本 students_new，查看内容：

```
# cat students new
2721010321 Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2721010409 Liwei   Sichuan tangwei 09/21/92  98  88  85  85  356  89
2721030227 wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
2821020115 Liumin  Henan   lixia   05/14/94  78  65  59  78  227  70
2821020225 Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
2821030215 Liyan   Sichuan Huli    06/30/93  75  65  55  45  240  60
2921010217 Luowei  Sichuan Hewei   07/05/92  90  80  78  81  329  82
2921020632 Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2921050313 Heli    Xizang  Tangwei 11/12/94  56  78  80  45  259  65
```

利用 cut 剪切时，剪切的文件可以来自文件，也可以来自管道的其他命令的输出等。完成剪切后，cut 与大多数文本操作命令一样，不会操作原有文本（即不改变原有文本的内容），而是直接将结果输出到标准输出。

cut 命令工作时，以行为单位剪切整个文本（即同样的剪切命令会对文本的每一行都生效）。

（1）对已经计算好字节数或具备特殊格式的文本进行剪切时，使用选项 b 按字节剪切是比较方便的。

例如要剪切所有学生的学号，学号具有特殊格式（每个学生的学号都是 10 个字节）：

```
#使用选项 b 表示剪切的单位字节，使用-10 表示剪切所有行的前 10 个字节
# cut -b-10 students_new
2721010321
2721010409
2721030227
……
```

（2）有时可能某个特殊字段的一小部分代表了某些特殊的含义（例如产地代码、标准号等）。当需要剪切这些具备特殊含义的部分时，可以使用选项 c 剪切某个字段的特定几个字符。例如要剪切出学生所在系和班级的代号：

```
#使用选项 c 表示按字符为单位剪切，5-8 表示每行的第 5～8 个字符
# cut -c5-8 students_new
0103
0104
0302
……
```

（3）大多数情况下都是对某一个或多个字段进行剪切，这时可以使用选项 f 对多字段进行剪切。例如要剪切出所有学生的学号、姓名和辅导员：

```
#使用选项 f 按字段剪切，使用 1-2,4 表示剪切第 1、2、4 个字段
# cut -f1-2,4 students_new
2721010321  Xuli     Luolei
2721010409  Liwei    tangwei
2721030227  wangtao Huli
……
```

（4）有时字段之间的分隔符不是制表符 Tab，这时应该使用选项 d 指定新的字段分隔符。例如要剪切 students4 文件中的前 3 个字段：

```
#使用选项 d 指定字段分隔符，并用 1-3 表示剪切前 3 个字段
# cut -d# -f1-3 students4
2821020225#Liulu#0
2821020116#Yangrui#B
2721010321#Xuli#0
……
```

一般情况下，都将 cut 命令的输出重定向到一个文件中，以便于执行粘贴操作。

注意： *在使用 cut 执行剪切操作时，无论是否需要，通常都将一行的标志性字段一起剪切（能唯一确定记录的字段），例如学号、序号等，以便于粘贴出问题时更容易查找。*

5.6.3　数据粘贴命令 paste

上一小节中讲解了如何使用 cut 从文本中剪切内容。通常完成剪切之后，需要将剪切的内容粘贴到另一个文本中。这时就需要使用数据粘贴命令 paste，其主要用来将两个相关性文本粘贴在一起形成一个新的文本。

执行粘贴之前，应该对粘贴的文本进行整理，确保两个文件每一行的文本内容之间具有一定的相关性。

【命令格式】

```
paste [option] [file...]
```

【常用选项】

- d：对新生成的文本指定新的字段分隔符，默认是制表符 Tab。
- s：将粘贴的内容合并成行（即横向粘贴）。

【用法示例】

先将上一小节中的文件 students_new 中的学号、姓名和总成绩利用 cut 命令剪切出来，分别保存到文件 paste1、paste2 和 paste3 中，本小节将利用这 3 个文件讲解粘贴操作。

（1）将学号和成绩粘贴在一起：

```
#直接使用粘贴的多个文本，paste 命令会直接将它们粘贴在一起
# paste paste1 paste3
```

```
2721010321      321
2721010409      356
2721030227      320
……
```

（2）如果要按指定的顺序粘贴，可以按先后顺序依次输入相关文件。例如想要先粘贴成绩再粘贴姓名：

```
#输入文件的顺序不同，产生的结果也不同
# paste paste3 paste2
321     Xuli
356     Liwei
320     wangtao
……
```

（3）paste 还可以进行横向粘贴，这在一些特殊情况下很有用。例如将学生姓名及成绩进行横向粘贴：

```
#使用选项 s 进行横向粘贴
# paste -s paste2 paste3
Xuli    Liwei   wangtao Liumin  Liulu   Liyan   Luowei  Xiayu   Heli
321     356     320     227     325     240     329     334     259
```

（4）使用 paste 进行粘贴时，使用的文件可以不止两个，即可以有多个文件同时进行粘贴。多个文件进行粘贴时，可以为每一个字段指定一个不同的分隔符。

例如要将 3 个文件粘贴起来，第 1、2 个字段之间仍旧用制表符分隔，而第 2、3 个字段之间用“#”分隔：

```
使用选项 d 指定第 1、2 个字段使用 Tab 制表符分隔，第 2、3 个字段使用“#”分隔
# paste -d'\t#' paste1 paste2 paste3
2721010321      Xuli#321
2721010409      Liwei#356
2721030227      wangtao#320
……
```

上面的命令中，paste 在进行粘贴时，依次使用指定的域分隔符，对粘贴后的域进行分隔。

（5）处理来自管道的文本时，paste 利用“-”代表来自管道的文本。例如要在文本 paste1 的每一行前面加上序号（序号已经保存在文件 ps 中）：

```
#在 paste 命令中使用“-”表示来自管道的文件
# cat paste1 | paste -d"#" ps -
1#2721010321
2#2721010409
3#2721030227
……
```

在上面的示例中，如果要交换序号和学号的位置，只需要交换参数减号“-”和文件 ps 的位置即可。

注意：在一些版本中，cut 和 paste 这两个命令会严格按字段分隔符对字段进行划分。例

如当有空格在制表符前后时，空格也会被当作一个字段。因此在剪切和粘贴的过程中应该反复尝试，直到输出结果正确后再重定向到文件中。

5.6.4　数据连接命令 join

学习过数据库的读者，可能对 join 连接并不陌生。join 连接主要用来将两个有一定关联性的表合并起来。此处要学习的 join 连接也是一样，主要用来将两个相关联的文件连接起来。

这两个文件中有一些字段是有关联的，例如前面介绍的两个示例文件：students_new 和 students2。这两个文件中，students_new 用于保存学生的基本信息和一些课程的成绩，而 students2 文件则用于保存学生在几门不同的实践课程中的成绩。这两个文件之间的关联性在于：每个文件的第 1 个字段都是学号，且每一个学生的学号是唯一的。像这种具有唯一性关联的文件，就可以使用 join 命令连接。

【命令格式】

```
join [option] file1 file2
```

【常用选项】

- a：用于输出两个文件中有关联的和没有关联的行，将有关联的行执行连接后输出，没有关联的行按预定的格式输出。
- e：在文件 1 和文件 2 中查找关联字段，如果没有关联字段，则将无关联的行相应的字段用参数指定的字符串替代。
- i：在连接过程中忽略大小写。
- j：使用指定的字段作为关联字段连接。
- o：格式化输出。
- t：设置字段间的分隔符，默认为空格或制表符 Tab。
- v：与选项 a 的作用相同，但选项 v 只输出无关联的行。

使用 join 命令时，join 用数字 1 来表示参数文件 1，用数字 2 来表示参数文件 2。

【用法示例】

在开始介绍 join 之前，将前面介绍的两个文件 students_new 和 students2 进行简单整理并排序。整理之后的名称为 students_join 和 students2_join，其内容如下：

```
#使用 cat 命令查看示例文件 students_join 的内容
# cat students_join
2721010321  Xuli    Jiangsu Luolei  12/25/92    76      81      85      79
2721010409  Liwei   Sichuan tangwei 09/21/92    98      88      85      85
2721030227  wangtao Yunnan  Huli    03/21/93    87      76      69      88
2821020115  Liumin  Henan   lixia   05/14/94    78      65      59      78
2821020225  Liulu   Sichuan Lixia   01/23/93    89      76      88      72
2821030215  Liyan   Sichuan Huli    06/30/93    75      65      55      45
2921010217  Luowei  Sichuan Hewei   07/05/92    90      80      78      81
2921020632  Xiayu   Shanxi  Hetao   03/26/93    78      86      92      78
2921050313  Heli    Xizang  Tangwei 11/12/94    56      78      80      45
#使用 cat 命令查看示例文件 students2_join 的内容
```

```
# cat students2_join
2721010321  Xuli       75
2721010409  Liwei      60
2821020116  Yangrui    75
2821020225  Liulu      85
2821030215  Liyan      75
2921020632  Xiayu      0
2921050313  Heli       85
```

（1）默认情况下 join 会使用参数文件的第 1 个字段作为关联字段。例如使用 join 将两个示例文件连接起来：

```
#使用 join 命令直接将示例文件执行连接
# join students_join students2_join
2721010321  Xuli   Jiangsu  Luolei  12/25/92    76  81  85  79  Xuli     75
2721010409  Liwei  Sichuan  tangwei 09/21/92    98  88  85  85  Liwei    60
2821020225  Liulu  Sichuan  Lixia   01/23/93    89  76  88  72  Liulu    85
2821030215  Liyan  Sichuan  Huli    06/30/93    75  65  55  45  Liyan    75
2921020632  Xiayu  Shanxi   Hetao   03/26/93    78  86  92  78  Xiayu    0
2921050313  Heli   Xizang   Tangwei 11/12/94    56  78  80  45  Heli     85
```

从上面的结果不难看出，直接使用 join 连接有以下问题。

- join 在进行连接时，没有输出两个文件无关联的记录。
- 由于默认输出时以空格作为字段分隔符，输出结果比较混乱。
- 输出的结果中，两个文件相关联的字段只输出一次。

（2）对于无关联字段的记录，有时要查看这些记录以便于作出判断或查找相关数据。这时可以使用选项 a 或 v 查看无关联的记录。

例如使用选项 a 查看两个示例文件中无关联的记录：

```
#使用参数 a1、a2 显示两个文件中的无关联记录
# join -a1 -a2 students_join students2_join
2721010321  Xuli    Jiangsu Luolei  12/25/92    76  81  85  79  Xuli     75
2721010409  Liwei   Sichuan tangwei 09/21/92    98  88  85  85  Liwei    60
2721030227  wangtao Yunnan  Huli    03/21/93    87  76  69  88
2821020115  Liumin  Henan   lixia   05/14/94    78  65  59  78
2821020116  Yangrui 75
2821020225  Liulu   Sichuan Lixia   01/23/93    89  76  88  72  Liulu    85
2821030215  Liyan   Sichuan Huli    06/30/93    75  65  55  45  Liyan    75
2921010217  Luowei  Sichuan Hewei   07/05/92    90  80  78  81
2921020632  Xiayu   Shanxi  Hetao   03/26/93    78  86  92  78  Xiayu    0
2921050313  Heli    Xizang  Tangwei 11/12/94    56  78  80  45  Heli     85
```

上面的命令中使用比较混乱的方式输出了合并后的结果，以及没有关联的记录。

（3）由于使用选项 a 显示得比较混乱，此时可以使用选项 v，只显示没有关联的记录：

```
#使用选项 v1、v2 只显示示例文件中没有关联的记录
# join -v1 -v2 students_join students2_join
2721030227  wangtao Yunnan  Huli    03/21/93    87  76  69  88
2821020115  Liumin  Henan   lixia   05/14/94    78  65  59  78
2821020116  Yangrui 75
2921010217  Luowei  Sichuan Hewei   07/05/92    90  80  78  81
```

（4）使用连接时，可能并不需要显示两个文件中的所有字段，例如文件 students_join 中已经包括了学生姓名，students2_join 中的姓名就不必再显示。此时可以使用选项 o 格式化输出，将不需要输出的字段屏蔽。

例如将文件 students_join 和 students2_join 连接之后重新输出，不输出用于检查和修正错误的字段：

```
#使用选项 o 指定连接后，只显示参数文件 1 中的域 1、2、4、6、7、8、9 和参数文件中的域 3
#然后使用 awk 命令规范命令输出，并统计总成绩和平均成绩
# join -o "1.1,1.2,1.6,1.7,1.8,1.9,2.3" students_join students2_join | awk
'{printf "%-15s %-10s %d %d %d %d %d %d %d\n", $1, $2, $3, $4, $5, $6, $7,
($3+ $4+ $5+ $6+ $7), ($3+$4+$5+$6+$7)/5}'
2721010321      Xuli        76   81   85   79   75   396    79
2721010409      Liwei       98   88   85   85   60   416    83
2821020225      Liulu       89   76   88   72   85   410    82
2821030215      Liyan       75   65   55   45   75   315    63
2921020632      Xiayu       78   86   92   78   0    334    66
2921050313      Heli        56   78   80   45   85   344    68
```

上面这个示例中，先用选项 o 将输出格式化，只输出第 1 个文件中的学号、姓名、4 门课程的成绩和第 2 个文件的实践课成绩。然后利用管道将结果交给 awk 命令处理，awk 使用 printf 函数将各个字段格式化输出，并重新计算总成绩和平均成绩，然后输出。

（5）有时两个文本相关联的字段并不在文件的第 1 个字段，这时应该使用选项 j 向 join 指定两个文件中的关联字段。

例如使用姓名作为关联字段进行连接：

```
使用选项 j 指定使用参数文件 1 的第 2 个字段和参数文件 2 的第 2 个字段作为关联字段
#  join  -j1  2  -j2  2  -o  "1.1,1.2,1.6,1.7,1.8,1.9,2.3"  students_join1
students2_join1
2921050313 Heli 56 78 80 45 85
2821020225 Liulu 89 76 88 72 85
2721010409 Liwei 98 88 85 85 60
……
```

（6）有时要连接的两个文件并非使用空格或制表符为字段分隔符，这时可以使用选项 t 指定文本的字段分隔符。例如连接下面两个文本：

```
#使用 cat 命令查看 a.txt 的内容
# cat a.txt
1#a#Z
2#b#Y
3#c#X
4#d#W
#使用 cat 命令查看 b.txt 的内容
# cat b.txt
1#kl#12
2#bd#75
3#lu#ds
7#ds#vc
#使用选项 t 指定连接使用的字段分隔符
# join -t"#" a.txt b.txt
```

```
1#a#Z#kl#12
2#b#Y#bd#75
3#c#X#lu#ds
```

（7）在对一些重要的数据进行连接时应该谨慎，有可能两个文件中的一些记录没有与之相关联的行。这种情况下连接时应该使用一些特殊的标记，对没有与之相关联的对应的相关字段进行标记，以便于在执行完连接操作后，将这些没有关联的记录进行补充或修改。可以使用选项 e 对这些记录进行标记。

为了更直观地查看选项 e 的作用，将上面的两个示例文件 a.txt 和 b.txt 的字段分隔符修改为空格。然后利用 join 命令执行连接操作，并用“#”标记没有与之相关联的对应的相关字段：

```
#查看文件 a1.txt 的内容
# cat a1.txt
1  a  Z
2  b  Y
3  c  X
4  d  W
#查看文件 b1.txt 的内容
# cat b1.txt
1  kl  12
2  bd  75
3  lu  ds
7  ds  vc
#使用选项 e 指定没有关联字段的填充符
# join -e "#" -a1 -a2 -o "1.1,1.2,2.1,2.2" a1.txt b1.txt
#注意命令输出中“#”出现的位置
1 a 1 kl
2 b 2 bd
3 c 3 lu
4 d # #
# # 7 ds
```

在上面的命令中，使用了选项 o 格式化输出的同时，还使用 a1 和 a2 选项将两个示例文件中无关联的记录也输出。与前面不同的是，使用选项 e 后，join 将原本无关联记录的字段用指定的字符串进行填充。

使用选项 e 指定填充字符时，必须使用选项 o 对输出进行格式化，否则选项 e 将无法找到要填充的字段位置。

注意：使用 join 进行连接时，应该先以两个文件的关联字段为关键字，进行排序操作，否则在进行连接时可能会出现一些意外的错误。

5.6.5　去除重复命令 uniq

命令 uniq 经常用于去除一个文本中的重复行。前面介绍过用 sort 命令去除重复行，与 uniq 命令相比，二者有较大的差别。用 sort 命令去除重复行时，sort 将整个文本中的多个重复的行去除，仅保留一行，而 uniq 会将多个连续重复的行去除，仅保留一行。以下面这

个文本为例：

```
#查看 stu_uniq 的内容
# cat stu_uniq
Liulu   Sichuan
Liwei   Sichuan
Liumin  Henan
Liumin  Henan
Liumin  Henan
Liumin  Henan
Xuli    Jiangsu
Liumin  Henan
Liulu   Sichuan
```

在上面这个文本中，有多个姓名为 Liumin 的学生信息。如果使用 sort 命令去除重复行，将会留下一行姓名为 Liumin 的学生信息；如果使用 uniq，则只会去除前 3 行重复的内容，原因在于后面的行虽然重复，但没有连续出现。

【命令格式】

```
uniq [option] [input[output]]
```

【常用选项】

- ❑ c：输出重复行的重复次数。
- ❑ d：仅输出重复的行。
- ❑ f：忽略一些字段，只比较指定的字段。
- ❑ i：忽略大小写。
- ❑ s：忽略一些字符，只比较指定的字符。
- ❑ u：输出不重复的行。
- ❑ w：指定要比较的字符位置。

【用法示例】

（1）对示例文件 stu_uniq 使用 uniq 命令并查看结果：

```
#使用 uniq 命令去掉文件 stu_uniq 中的重复行
# uniq stu_uniq
Liulu   Sichuan
Liwei   Sichuan
Liumin  Henan
Xuli    Jiangsu
Liumin  Henan
Xiayu   Shanxi
Liulu   Sichuan
```

从上面的结果可以看出，uniq 命令仅仅去除了姓名为 Liumin 的多条连续重复的行。

（2）可以利用选项 c 和选项 d 输出重复的行信息：

```
#使用选项 c 统计重复行的次数
# uniq -c stu_uniq
      1 Liulu   Sichuan
      1 Liwei   Sichuan
      4 Liumin  Henan
```

```
     1 Xuli    Jiangsu
……
#使用选项 d 输出重复行的内容
# uniq -d stu_uniq
Liumin  Henan
```

从上面的结果可以看出，uniq 在计算重复的行时，仅仅计算了连续重复的行，因此在对文件使用 uniq 命令之前，应该对文件进行排序。

（3）输出文件中没有重复的行可以使用选项 u：

```
#使用选项 u 输出文件中没有重复的行
# uniq -u stu_uniq
Liulu       Sichuan
Liwei       Sichuan
Xuli        Jiangsu
……
```

（4）这里引入一个新的文本文件 char，内容如下：

```
#查看示例文件 char 的内容
# cat char
aaaaa   bbbAA
aaaaa   bbbAA
aaaaa   cbbAA
bbaaa   abbAA
BDBBB   AAABB
BCBBB   AAABB
ABBBB   AACBB
```

利用 uniq 去除重复的行时，uniq 总是从第 1 个字段到最后一个字段进行一一对比。然而有时可能只需要对几个字段或几个字符进行对比，这时可以结合选项 f 和选项 s 去除这些重复的行：

```
#使用选项 f 忽略第 1 个字段，然后使用 s3 从第 2 个字段的第 3 个字符处开始对比
# uniq -f1 -s3 char
aaaaa   bbbAA
BDBBB   AAABB
ABBBB   AACBB
```

上面这条命令先忽略第 1 个字段，从第 2 个字段的第 3 个字符开始去除重复的行，这时第 2 个字段的第 3 个字符后面内容重复的行没有被输出。

注意：在使用 uniq 命令去除重复行时，某些系统可能出现个别选项无法使用或结果不同的情况，此时应该阅读相关命令的手册。

5.6.6　分割文件命令 split

在管理和维护 Linux 的过程中，有时可能为了传送文件方便，或运用其他工具时缓冲区限制等因素，无法使用大文件。此时可以利用 Linux 自带的命令 split，将很大的文件分割成若干个小文件。本小节将简单介绍分割文件命令 split 的用法。

【命令格式】

```
split [option] [input file] [output file]
```

【常用选项】

- ❑ l：按行对文件进行分割。
- ❑ b：按字节对文件进行分割。
- ❑ C：按字节对文件进行分割，split 会尽量保持一个整行。
- ❑ d：使用数字作为输出文件的后缀。

在指定输出文件名称后，split 会将分割的小文件以指定的名称为前缀，在其后加上诸如 aa、ab 等后缀作为分割后的文件名。

【用法示例】

（1）现有一个文件 student，其中共有 7 行内容，要将这个文件按每 3 行放入一个文件的办法分割文件：

```
#查看文件 student 的内容
# cat student
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2721010409  Liwei   Sichuan tangwei 09/21/92  98  88  85  85  356  89
2721030227  wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
2821020115  Liumin  Henan   lixia   05/14/94  78  65  59  78  227  70
2821020225  Liulu   Sichuan Lixia   01/23/93  89  76  88  72  325  81
2921020632  Xiayu   Shanxi  Hetao   03/26/93  78  86  92  78  334  84
2921050313  Heli    Xizang  Tangwei 11/12/94  56  78  80  45  259  65
#使用选项 l 指定每 3 行划分为一个文件，并将划分后的文件命名为 student
# split -l 3 student student
#查看分割后的文件
# ls -l
-rw-r--r-- 1 root root  198 Aug 12 11:30 studentaa
-rw-r--r-- 1 root root  193 Aug 12 11:30 studentab
-rw-r--r-- 1 root root   65 Aug 12 11:30 studentac
#验证分割后文件的内容
# cat studentaa
2721010321  Xuli    Jiangsu Luolei  12/25/92  76  81  85  79  321  80
2721010409  Liwei   Sichuan tangwei 09/21/92  98  88  85  85  356  89
2721030227  wangtao Yunnan  Huli    03/21/93  87  76  69  88  320  80
```

上面这条命令使用 split 将文件 student 按每 3 行分割成 1 个文件的办法，将文件分割成 studentaa、studentab 和 studentac 3 个文件。

（2）出于多种原因，文件传送工具可能会对传送的文件大小有限制（例如电子邮件附件），也可能文件太大超过了编辑工具临时缓冲区的大小等。这时可以用选项 b 指示 split 按字节分割文件。例如：

```
#使用选项 b 按字节分割文件
# split -b 1024 smb.conf smb
```

利用 split 对文件按字节进行分割时，split 每一次读取 512 个字节的数据放入分割的文件中，然后进行下一次读取数据操作。因此通常推荐分割后的文件大小，应该为 512 字节的倍数。

注意：在使用 split 命令分割文件时，某些系统可能会对分割的行数或字节数有所限制，此时应该先阅读相关文档，再进行分割。

5.7 小　　结

- 5.1 节中介绍了用于查找文件的 find 命令。该命令既可以查找某一个具体的文件，也可以查找具有某一类特征的文件。这个使用简单且功能十分强大的工具，在管理和维护 Linux 系统的过程中经常用到。通常的用法是查找某一时间内更改过的文件，以便备份文件，在多用户系统上也经常用来查看具有某种权限的文件等。
- 5.2 节中讲解了用于从文本中抽取文本的命令 grep。该命令经常用于从其他命令的输出中抽取一个匹配的行，以便于查看系统运行状态等。
- 5.3 节中介绍了 sed 流编辑器。sed 虽然是一个非交互式的流编辑器，但功能十分强大，在编写一些系统管理和自动初始化脚本时，经常用来替换一些配置文本。
- 5.4 节主要介绍了 awk 命令及其用法。对于大多数初学者而言，这是一个非常棘手的工具。但就其强大的功能而言，这绝对是一个十分值得学习的工具。awk 在系统管理和维护的过程中，经常用来从文本和命令输出中抽取一些数据。
- 5.5 节介绍了本章中最简单的命令 tr，相比其他几个命令，这个命令的用法最为简单。使用这个命令可以轻松地对字符串进行大小写转换、删除重复字符等。
- 5.6 节中讲解了用于文本合并和分割的几个小命令。虽然这些工具并不是很常用，但在统计数据等方面利用这些工具非常方便。

Linux 系统管理员和使用者，经常需要从一些文本文件或命令输出中查找和筛选一些关键数据。从这些数据中可以判断出系统当前或曾经运行的情况，可以利用这些数据编写一个自动化脚本维护整个业务系统的稳定运行，还可以利用这些数据制作一个报表等。因此应该掌握本章中的大多数命令的用法，以便于需要时能够使用这些命令。

第 6 章　用户和文件权限管理

Linux 作为一个多用户、多任务系统，可能同一时间有许多用户登录并使用系统。作为管理员，首先需要考虑如何有效地管理系统中的用户，其次要考虑如何运用文件权限让系统中的多个用户同时使用系统，但不互相影响。本章将介绍 Linux 系统中用户和文件权限的管理。

本章主要涉及的知识点如下。

- 简单介绍系统用户文件，添加、删除及管理系统中的用户。
- 系统用户组文件介绍，添加、删除及管理用户组。
- 文件属主、属组及文件基本权限系统概述。
- Linux 系统中的权限管理命令介绍。
- 权限掩码 umask，suid、sgid 和 sticky 权限介绍。
- POSIX ACL 权限系统及管理命令介绍。

6.1　用户管理

对于多用户、多任务系统，对系统中的用户进行管理，是管理员的一项基本工作。针对用户的管理内容包括添加、删除用户，禁止用户登录等，本节将简单介绍 Linux 系统中的用户管理。

6.1.1　系统用户文件概述

许多读者都知道，在 Linux 系统中一切都是文件，包括硬盘、键盘等许多设备都是一个文件。不例外的是，系统中的用户也是保存在文件中的。与系统用户一起保存在文件中的还有用户的设置，例如用户的家目录、使用的 Shell、用户的密码等。

由于学习用户管理命令必然要涉及系统用户文件和用户密码文件，因此在讲解用户管理命令之前，先简单介绍 Linux 系统中的系统用户文件和用户密码文件。

1. 系统用户文件

用于保存系统用户及用户设置的文件是/etc/passwd。查看系统用户文件的内容：

```
#使用 cat 命令查看系统用户文件的内容
# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
```

```
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
……
```

仔细查看系统用户文件的每一行，不难发现每一行都使用冒号“:”将各个字段分隔开。这些字段就是用户的设置，从左起这些字段的含义如下（以第 1 行 root 用户配置行为例作说明）。

- 第 1 个字段为用户名。例如文件第 1 行中的 root 就表示根用户。
- 第 2 个字段为用户的登录密码。在早期系统中，这个字段用于保存密码，此处的字母 x 表示密码保存在用户密码文件/etc/shadow 中。
- 第 3 个字段是用户的 ID 号（即 UID）。本例中 root 用户对应的 UID 号为 0。
- 第 4 个字段用于标识用户所属组的 ID 号（即 GID）。本例中 root 用户对应的 GID 号为 0。
- 第 5 个字段是可选的，通常为空（因此在系统用户文件中，许多行都使用了两个冒号）。这个字段一般用于保存用户的其他信息（例如全名）等。本例中 root 用户的该字段为 root。
- 第 6 个字段是用户的家目录。家目录主要用于存放用户的初始化文件（初始化文件用于初始化用户环境、启用软件个性设置等）、个人文件等。本例中 root 用户的家目录为/root。
- 第 7 个字段保存的是用户使用的 Shell。在本例中 root 用户使用的 Shell 是/bin/bash。

小知识： 在 Linux 系统中，UID 号从 0～499 为系统保留给系统用户（通常是软件的专有用户）使用的，因此管理员添加的用户 ID 号通常大于 500。

2. 用户密码文件

用于保存密码的文件是/etc/shadow，由于单词 shadow 有影子的意思，因此人们也将这个文件称为影子文件。查看影子文件的内容：

```
#使用 cat 命令查看影子文件的内容
# cat /etc/shadow
root:$1$QsYRjZye$ek0.rDuRH4QyI0Cu.mFDE0:14789:0:99999:7:::
bin:*:14782:0:99999:7:::
daemon:*:14782:0:99999:7:::
adm:*:14782:0:99999:7:::
lp:*:14782:0:99999:7:::
sync:*:14782:0:99999:7:::
shutdown:*:14782:0:99999:7:::
halt:*:14782:0:99999:7:::
mail:*:14782:0:99999:7:::
……
```

密码文件 shadow 也是一个格式化文件，同样用冒号分隔各个字段。从左起这些字段的含义如下（以文件第 1 行 root 用户为例作说明）。

- 第 1 个字段为用户名。例如本例中的 root 表示 root 用户。
- 第 2 个字段用于保存加密后的密文字符串。如果该字段以“1”开头，表示密码采用 MD5 加密（MD5 是一种常见的加密算法，经常用于加密密码等不可逆的字符串）。本例中以1开头的一长串字符，就表示使用 MD5 加密后的密文。
- 第 3 个字段表示最后一次密码修改的天数（从 1970 年 1 月 1 日算起）。
- 第 4 个字段表示两次修改密码之间的间隔天数，在此期间用户不能修改密码（0 表示可以随时修改密码）。本例中的 0 表示 root 用户随时可以更改自己的密码。
- 第 5 个字段表示密码最长存留的天数（1 表示用户不能修改密码）。本例中的 99999 可以保证 root 用户的密码存留足够的时间。
- 第 6 个字段用于标识密码过期之前的警告天数（–1 表示没有警告）。本例中的警告天数为 7 天。
- 第 7 个字段表示密码过期之后禁止用户登录的天数（1 表示不会被禁用）。本例中为空表示 root 用户不会被禁止登录。
- 第 8 个字段表示过期后禁用的天数（–1 表示不会被禁用），以自 1970 年 1 月 1 日以来的天数表示。本例中为空表示 root 用户不会被禁用。
- 最后一个字段为保留字段，该字段一般都为空。

系统用户文件和密码文件中保存的用户名及密码等都由系统自动生成，并且其中保存的用户大部分都是软件专有用户。软件专有用户是一类比较特殊的用户，这类用户通常是为了保证系统安全运行而存在的，不能直接登录系统。

以软件专有用户的身份运行软件，可以大大降低系统被攻击的风险。例如以用户 mysql 的身份运行 mysqld 服务，即使 mysqld 服务被黑客攻破，也不会影响到系统中的其他文件。除非管理员修改，默认情况下 mysql 用户无权删除和修改文件系统中的其他文件。关于使用软件专有用户避免系统被攻击的详细内容，请参考相关安全文档和说明。

提示： 本小节中介绍的系统用户文件和用户密码文件都由系统自动生成，一般不要使用编辑文件的方式修改相关设置。

6.1.2　添加用户命令 useradd

在多用户系统中，一个常见的情形是：新同事报到时，可能需要我们向系统中添加相应的用户，以便于新同事能使用系统中的资源。即便在单用户系统中，有时也需要为安装的软件添加专有用户。例如编译安装的数据库软件 MySQL 时，需要手动使用命令添加其专有用户 mysql。

从上面的情形可以看出用户管理在 Linux 中的重要性。本小节将简单介绍如何使用 useradd 和 adduser 命令添加用户。

查看这两个命令的文件：

```
#使用 ls 命令查看 adduser 命令的命令文件
# ls -l /usr/sbin/adduser /usr/sbin/useradd
```

```
lrwxrwxrwx  1 root root     7 Sep  8 22:36 /usr/sbin/adduser -> useradd
-rwxr-x---      1 root root 72964 Aug 26  2008 /usr/sbin/useradd
```

从上面的命令输出中可以看出，命令 adduser 实际上是 useradd 的软链接文件，因此这两个命令的用法相同。

【命令格式】

```
useradd [option] username
```

【常用选项】

- g：指定新用户所属的用户组。
- s：指定新用户使用的 Shell。

除此之外还有一些其他的选项，感兴趣的读者可以阅读相关文档。

【用法示例】

（1）例如要创建一个名为 ljx 的用户：

```
#使用 useradd 命令添加新用户 ljx
# useradd ljx
#查看系统用户文件和影子文件验证用户是否添加成功
# cat /etc/passwd | grep ljx
ljx:x:501:501::/home/ljx:/bin/bash
# cat /etc/shadow | grep ljx
ljx:!!:14860:0:99999:7:::
```

从上面的命令输出中可以看出，在系统用户文件和用户密码文件中，都新增了一条以 ljx 开头的记录。

在系统中添加用户需要注意以下几点。

- 由于用户名会放入系统用户文件，因此不要使用中文用户名。任何时候都不要向系统配置文件中写入中文，这可能会导致一些意外的错误，即使写入的是一条注释信息。
- 任何时候都不建议在 Linux 系统中使用公共账号，这可能会引发一些问题。例如两个来自不同位置的同一账号，同时修改一个文件时，就可能引发一些冲突。
- 如果使用系统的人数较少，可以使用人名拼音的缩写作为用户名（本书中大部分地方如此），如果人数较多，建议使用人名拼音作为用户名。
- 虽然 Linux 系统中的用户名和用户组名都区分大小写，但建议不要使用大写字母。
- 用户名的唯一要求是方便系统中的其他用户识别，因此一般不要在用户名中使用除小写字母以外的特殊字符。

（2）有些用户可能更熟悉其他 Shell，这时可以使用选项 s 为其指定特殊的 Shell。例如新建一个名为 wlh 的用户，并更改其默认的 Shell 为 Ksh：

```
#使用选项 s 指定新建用户的默认 Shell
# useradd -s /bin/ksh wlh
#通过查看系统用户文件的方法验证设置
# cat /etc/passwd | grep wlh
wlh:x:502:502::/home/wlh:/bin/ksh
```

（3）添加新用户时，系统会自动分配新用户的家目录。默认情况下，家目录位于/home

目录中，并以用户名作为家目录的名称，查看用户的家目录：

```
#使用 ls 命令查看/home 目录中的用户家目录
# ls -l /home/
total 40
drwx------ 3 ljx        ljx     4096    Sep  9 02:53 ljx
drwx------ 2 root       root    16384   Sep  8 22:31 lost+found
drwx------ 3 user1      user1   4096    Sep  9 02:26 user1
drwx------ 3 wlh        wlh     4096    Sep  9 03:04 wlh
```

（4）分配新用户的家目录后，系统还会添加新用户的初始化文件（初始化文件通常用于初始化用户的系统、软件环境，这些内容将在本书的第 16 章中介绍）。通常用户的初始化文件被放在家目录中，并以隐藏文件的形式存在。查看用户的初始化文件的命令如下：

```
#使用 ls 命令查看家目录中的用户初始化文件
# ls -al /home/ljx/
total 56
……
#以下以点号开头的文件都是用户的初始化文件
-rw-r--r-- 1 ljx  ljx   33 Sep  9 02:53 .bash_logout
-rw-r--r-- 1 ljx  ljx  176 Sep  9 02:53 .bash_profile
-rw-r--r-- 1 ljx  ljx  124 Sep  9 02:53 .bashrc
……
```

除了用户家目录中的初始化文件之外，命令还会在系统邮件目录/var/spool/mail 中，为用户添加邮件文件，以便于用户接收邮件。

（5）根据安装的软件包不同，家目录中可能还会存在一些别的初始化文件，例如编辑器 Vim 的初始化文件.viminfo 等。这些初始化文件的模板被放置在目录/etc/skel 中。查看用户初始化的模板文件的命令如下：

```
#查看系统用户初始化模板文件
# ls -al /etc/skel/
```

关于这些初始化文件的内容和使用方法，将在本书的第 16 章中介绍。

（6）编译安装软件时，有些软件会要求建立软件专有用户。由于软件专有用户不能直接登录系统，因此需要使用选项 s 修改专有用户的 Shell。例如使用 useradd 命令添加 MySQL 的专有用户：

```
#使用选项 s 修改用户使用的 Shell 为/sbin/nologin，以阻止其登录
# useradd -s /sbin/nologin mysql
```

由于软件专有用户不能登录系统，因此通常不需要为其设置密码。

虽然系统会禁止空密码用户登录，但仍然建议修改软件专有用户的 Shell，以免存在可能的安全隐患。

注意：/sbin/nologin 是一个特殊的 Shell，系统会禁止使用此 Shell 的用户登录系统。

（7）有时可能需要在建立用户时指定用户的用户组，此时可以使用选项 g。例如：

```
#使用选项 g 指定添加的新用户 user2 的用户组为 user1
# useradd -g user1 user2
```

6.1.3　设置用户密码命令 passwd

为系统添加用户后，如果不设置密码，大多数发行版都会禁止用户登录系统，因此添加用户后的首要任务是为用户设置密码。设置密码可以使用 passwd 命令，本小节将简单介绍设置用户密码命令 passwd 的用法。

【命令格式】

```
passwd [username]
```

使用 passwd 命令为用户设置密码时，通常不需要使用选项。

使用 passwd 修改用户密码需要注意以下几点。

- ❑ 如果没有为 passwd 命令指明用户，passwd 将会修改当前用户的密码。
- ❑ passwd 命令并非只有 root 用户才能使用，普通用户也可以使用该命令修改自己的登录密码。
- ❑ 修改密码时，应该注意新设置密码的安全性（例如密码要有一定的复杂性，不要使用生日、电话号码等容易破解的密码）。

【用法示例】

新创建的用户都没有设置密码，为了用户能正常使用系统，通常在用户被创建后就需要立即为其创建密码。

（1）为新创建的用户 ljx 设置密码：

```
#使用 passwd 命令为用户 ljx 创建密码
#只有 root 用户才能设置其他用户的密码
# passwd ljx
Changing password for user ljx.
#命令要求用户输入新密码
#注意此处输入的密码与登录系统一样，都是不可见的，输入完成后直接按 Enter 键即可
New UNIX password:
#重复输入密码
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
#使用查看影子文件的方法验证
# cat /etc/shadow | grep ljx
ljx:$1$/y20cAi8$ckzb1kX1T/CDYjuTaVGW6.:14860:0:99999:7:::
```

此处输入的密码与登录时输入的密码一样是不可见的，输入完成后按 Enter 键即可结束输入。对比文件内容可以看出，影子文件使用两个感叹号“!!”表示空密码。

小知识： Linux 系统为不同的用户使用了不同密钥，因此即使两个用户使用相同的密码，影子文件中保存的密文也会不同。

（2）如果使用 passwd 命令时，没有指定要修改的用户名称，passwd 将默认修改当前用户的密码。例如：

```
#如果没有指明修改密码的用户，passwd 命令默认修改当前用户
# passwd
#从以下提示可以看出当前修改的是 root 用户的密码
```

```
Changing password for user root.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

由于使用 passwd 命令不需要 root 用户权限，因此普通用户只能使用该命令修改自己的登录密码。

6.1.4　删除用户命令 userdel

当用户不再需要使用系统时（可能的原因是该员工已经离职），管理员为了系统的安全，需要删除系统中的用户，这时可以使用命令 userdel。

【命令格式】

```
userdel [option] username
```

【常用选项】

删除用户命令 userdel 常用的选项只有 r，该选项的功能是删除用户时，连同用户的家目录和邮件文件一并删除。

【用法示例】

（1）直接将要删除的用户名作为 userdel 命令的参数即可删除用户。例如要删除用户 bgh：

```
#使用 userdel 删除用户 bgh
# userdel bgh
```

使用上述命令后，系统将从系统用户文件和影子文件中移除该用户，但需要管理员手动删除用户的家目录和邮件文件。

（2）如果要连同用户的家目录和邮件文件一并删除，可以配合使用选项 r。例如删除用户 bgh 并删除其家目录和邮件文件：

```
#使用选项 r 删除用户及其家目录和邮件文件
# userdel -r bgh
# ls /home/
ljx  user1  wlh
```

当用户不再使用系统时，通常推荐将该用户的账号禁用一段时间后（禁用账号将在下一小节中介绍），再删除用户的家目录、个人文件及邮件文件，以避免丢失有用的数据。

小知识： 由于系统 umask 值（将在第 6.3.5 小节中介绍）的缘故，普通用户的文件一般只存放在其家目录内。除非 root 用户更改其 umask 值，或修改了某些目录的权限。

6.1.5　用户管理命令 usermod

用户管理的基本内容是禁用、启用用户账号（即锁定、解除锁定），以及修改用户账

号的某些属性等。在 Linux 系统中用户管理通常使用命令 usermod，本小节将简单介绍如何使用 usermod 命令管理系统用户。

【命令格式】

```
usermod [option] username
```

【常用选项】

- L：锁定用户，即禁止用户登录系统。
- U：解除锁定。
- e：指定用户过期的日期。
- f：指定用户过期之后的缓冲时间，即过期之后还能够登录的天数。
- d：为用户指定新的家目录。
- m：为用户指定新的家目录时，移动原来家目录中的所有文件。
- s：修改用户的默认 Shell。

除此之外，usermod 还有一些用于修改用户组的选项，这些选项将在用户组管理相关内容中介绍。

【用法示例】

（1）用户管理最常见的任务是锁定用户。例如要锁定用户 ljx：

```
#使用 usermod 的选项 L 锁定用户 ljx
# usermod -L ljx
# cat /etc/shadow | grep ljx
ljx:!$1$/y20cAi8$ckzb1kX1T/CDYjuTaVGW6.:14860:0:99999:7:::
```

（2）要解除已经锁定的用户可以使用选项 U。例如解除已锁定的用户 ljx：

```
#使用选项 U 解除已锁定的用户
# usermod -U ljx
# cat /etc/shadow | grep ljx
ljx:$1$/y20cAi8$ckzb1kX1T/CDYjuTaVGW6.:14860:0:99999:7:::
```

从上面两条命令执行后影子文件的密码字符串变化可以看出，锁定用户其实就是在影子文件中的用户密码字符串前加上感叹号“!”。

（3）要指定用户过期的时间，可以使用选项 e。例如设置用户 ljx 的过期时间为 2010 年 1 月 1 日：

```
#使用选项 e 指定用户 ljx 的过期时间
# usermod -e 2010-10-1 ljx
```

（4）设置用户过期时间之后，还需要使用选项 f 设置其可以登录系统的天数。例如设置用户 ljx 过期之后，还可以使用系统的时间为 5 天：

```
#使用选项 f 设置用户 ljx 过期后还可以使用系统的天数
# usermod -f 5 ljx
# cat /etc/shadow | grep ljx
ljx:$1$/y20cAi8$ckzb1kX1T/CDYjuTaVGW6.:14860:0:99999:7:5:14883:
```

（5）有时用户使用的家目录空间太小，需要重新为用户指定新的家目录，这时可以配合使用选项 d。例如要改变用户 ljx 的家目录到/file/user/home/ljx：

```
#使用选项 d 修改用户的家目录并验证设置
# usermod -d /file/user/home/ljx ljx
# cat /etc/passwd | grep ljx
ljx:x:501:501::/file/user/home/ljx:/bin/bash
# ls -al /file/user/home/ljx
total 16
drwxr-xr-x 2 root root 4096 Sep 10 03:37 .
drwxr-xr-x 3 root root 4096 Sep 10 03:37 ..
```

从上面的示例命令中可以看出，命令 usermod 虽然改变了用户的家目录，但并没有将家目录中的文件移动到新目录中。

（6）要连同家目录中的文件一并移动，可以配合使用选项 m。例如：

```
#使用选项 d 修改家目录的同时，用选项 m 移动家目录中的文件
# usermod -d /file/user/home/ljx/ -m ljx
# ls -al /file/user/home/ljx
total 56
……
-rw-r--r-- 1 ljx  ljx    33 Sep 10 03:58 .bash logout
-rw-r--r-- 1 ljx  ljx   176 Sep 10 03:58 .bash profile
……
```

提示：也可以通过修改文件/etc/login.defs 的方式，修改用户的密码策略。这个文件比较简单，此处不做详细介绍，感兴趣的读者可以在学习了本章内容之后，阅读相关文档了解这个配置文件。

6.2　用户组管理

Linux 系统中的用户组与 Windows 系统中的用户组类似：管理员可以创建多个用户组（以下简称组），并将多个用户加入一个组以实现权限集中管理。与 Windows 系统不同的是，Linux 系统创建用户时，系统会自动创建一个与用户名相同的用户组（与用户一起被创建的同名用户组称为私有组），并将用户加入该组。而在 Windows 系统中，系统会将新建的用户加入到一个名为 user 的组中。

许多应用系统都使用用户组作为权限实施的对象（例如文件服务器、FTP 服务器等），因此初学者必须对用户组有一定的了解。本节将简单介绍 Linux 系统中的用户组管理。

6.2.1　用户组文件概述

与 Linux 系统中的用户一样，用户组也将相关设置保存在文件中。用于保存用户组的文件是系统用户组文件/etc/group 和用户组密码文件/etc/gshadow。与用户密码文件类似，有时也将用户组密码文件称为用户组影子文件。

由于用户组管理必然要涉及这两个文件，因此在学习组管理之前，本小节先简单介绍系统用户组文件和用户组影子文件。

1．系统用户组文件

Linux 系统将用户组的相关设置保存在用户组文件/etc/group 中，查看该文件的内容如下：

```
#使用 tail 显示用户组文件的最后几行
# tail /etc/group
dbus:x:81:
haldaemon:x:68:
avahi:x:70:
……
user1:x:500:
ljx:x:501:
wlh:x:502:
```

用户组文件与系统用户文件、密码文件一样，也使用冒号分隔各个字段。从左起各字段的含义如下（以最后一行为例）。

- 第 1 个字段为用户组的名称。本例中的用户组为 wlh。
- 第 2 个字段是组密码，这里用于存放加密后的密码字符串。如果密码为空，表示不需要密码。
- 第 3 个字段是 GID，即用户组的 ID 号。本例中的用户组 ID 是 502。
- 第 4 个字段是用户列表，如果存在多个用户，可以使用逗号分隔。

在本例中，用户组的密码为 x，表示用户组的密码保存在用户组密码文件/etc/gshadow 中。

2．用户组密码文件

用户组密码文件用于保存用户组密码、用户组管理员等设置。查看用户组密码文件的内容如下：

```
#使用 cat 命令查看用户组影子文件
# cat /etc/gshadow
root:::root
bin:::root,bin,daemon
daemon:::root,bin,daemon
sys:::root,bin,adm
……
```

用户组影子文件也使用冒号分隔各字段，从左起这些字段的含义如下（以第 1 行为例）。

- 第 1 个字段为组名称。本例中的用户组名称为 root。
- 第 2 个字段用户组密码（加密后的密码字符串）。本例中为空。
- 第 3 个字段为用户组的管理员。如果有多个管理员，则使用逗号“,”分隔。本例中的管理员为空。
- 第 4 个字段为组成员列表。如果有多个成员，则使用逗号“,”分隔。

系统用户组文件和用户组影子文件都是由系统自动生成的，不建议使用手动修改的方法管理用户组，以免带来不必要的麻烦。

提示： 由于用户组密码、管理员等相关内容应用很少，因此本书将不介绍此部分内容，感兴趣的读者可以阅读相关文档。

6.2.2　添加用户组命令 groupadd

在 Linux 系统中实施权限细则时，通常将权限相似的用户添加到一个组，最后对组实施权限细则(例如允许某个用户组访问某个目录和其中的文件等)，以达到方便管理的目的。为此需要在系统中添加用户组，添加用户组可以使用命令 groupadd。

【命令格式】

```
groupadd groupname
```

添加用户组命令 groupadd 没有常用的选项，直接将组名作为其参数即可添加。

【用法示例】

（1）要添加一个名为 teacher 的组：

```
#使用 group 命令添加一个名为 teacher 的组
# groupadd teacher
#使用 tail 命令查看新添加的组
# tail /etc/group
……
user1:x:500:
ljx:x:501:
wlh:x:502:
teacher:x:503:
```

添加用户组时，应该为用户组取一个意义明显的名称，例如 admin（管理员）、teacher（教师）、student（学生）等。与用户一样，用户组名也建议使用中文、大写字母、特殊字符等。

（2）使用命令 groupadd 添加用户组后，创建新用户时就可以使用 useradd 的选项 g：

```
#使用选项 g 将新添加的用户添加到 teacher 中
# useradd -g teacher wj
```

6.2.3　删除用户组命令 groupdel

有时可能不再需要某一个用户组（例如某个部门被撤销、服务器将作它用时），这时需要将用户组从系统中删除。删除用户组可以使用命令 groupdel。

删除用户组命令 groupdel 没有常用的选项，直接为其指定要删除的用户组名即可。例如要删除用户组 teacher：

```
#使用 groupdel 删除用户组 teacher 并验证
# groupdel teacher
# tail /etc/group
……
user1:x:500:
ljx:x:501:
wlh:x:502:
```

6.2.4　用户组管理

用户组管理的主要内容是将用户加入不同的组，或从用户组中移除用户，以便使用 ACL（ACL 将在 6.4 节中介绍）以组为单位实施权限细则（当管理的用户较多时，不推荐对单个用户实施权限细则）。管理用户组主要使用命令 usermod，这个命令在 6.1 节中已经介绍过，本小节将继续介绍如何使用 usermod 管理用户组。

提示：通常将系统用户文件中指定的用户组称为用户的私有组，在用户组文件中为用户指定的组称为用户的附加组。当用户创建文件时，系统将使用用户的私有组作为文件的属组。

【用户组管理的常用选项】

- g：将用户的私有组改变为选项指定的组。
- G：为用户添加多个附加组，使用逗号作为分隔符。
- a：将用户以追加的方式添加到一个附加组。

除此之外 usermod 命令还有很多选项，读者可以阅读 6.1 节用户管理和相关文档。

【用法示例】

（1）要更改用户的私有组，可以配合使用选项 g。例如要改变用户 ljx 的私有组为 teacher：

```
#使用选项 g 修改用户 ljx 的私有组并查看相关文件验证
# usermod -g teacher ljx
# cat /etc/passwd | grep ljx
ljx:x:501:503::/file/user/home/ljx/:/bin/bash
# cat /etc/group | grep teacher
teacher:x:503:
```

（2）要为用户附加多个组，应该配合使用选项 G。例如要为用户 wlh 附加 teacher 和 admin 组：

```
#使用选项 G 将用户加入多个用户组并查看相关文件验证
# usermod -G teacher,admin wlh
# cat /etc/passwd | grep wlh
wlh:x:502:502::/home/wlh:/bin/bash
# tail /etc/group
……
wlh:x:502:
teacher:x:503:wlh
admin:x:504:wlh
teacher1:x:505:
```

（3）上面已经将用户 wlh 加入 teacher 和 admin 组，如果还需要将其加入用户组 teacher1，应该使用选项 a：

```
#使用选项 a 将用户 wlh 追加到组 teacher1 中
# usermod -aG teacher1 wlh
# tail /etc/group
……
```

```
wlh:x:502:
teacher:x:503:wlh
admin:x:504:wlh
teacher1:x:505:wlh
```

6.3　基本权限及管理命令

文件权限系统是 Linux 保证数据安全性的重要手段之一。与 Windows 相比，Linux 系统默认权限设置比较严格。但这并不能满足所有用户的需求，因此每个 Linux 用户都需要学习文件权限管理，以便使用文件权限保护自己的文件、隐私不被泄露。

Linux 的文件权限系统基本沿用 UNIX 的权限系统，用户可以利用这个简单的权限系统，设置文件的访问权限。用户可以利用文件的访问权限保护自己的文件及重要数据不被泄露。本节将简单介绍 Linux 系统中的文件权限管理。

6.3.1　文件的属主和属组

当文件被创建时，系统会以创建此文件的用户及用户所在的私有组为此文件添加属主和属组信息，并设置其默认访问权限。属主在 Windows 系统中通常称为文件所有者。文件属主可以对文件执行一切操作，包括读取、修改和删除等。

提示： 在 Linux、UNIX 系统中，root 用户拥有最高权限，因此可以读取、修改和删除系统中的任何文件。

创建一个新文件并以长格式查看文件信息：

```
#使用 touch 命令创建文件 a
$ touch a
#使用 ls -l 命令以长格式查看文件的详细信息
$ ls -l a
-rw-rw-r-- 1 wlh wlh 0 Sep 13 15:35 a
```

此处文件 a 的属主和属组都是 wlh（第 1 个 wlh 是属主，第 2 个为属组）。

默认情况下，系统会分给用户所在的组成员一些权限（以属组权限的方式体现），如果用户是多个组的成员，那么系统会使用用户文件/etc/passwd 中保存的组（即私有组）。

6.3.2　修改文件属主和属组命令 chown、chgrp

当文件被创建时，属主和属组就已经被添加，但有时需要改变文件的属主和属组。一个典型的例子：编译安装 MySQL 时（通常编译安装都会自定义数据目录），需要手动创建 MySQL 的数据目录，创建完成后还需要将数据目录的属主和属组修改为 mysql，这样数据库服务 mysqld 运行时才能读取、写入数据。

在 Linux 系统中，修改文件的属主和属组可以使用命令 chown 和 chgrp。本小节将简单介绍这两个命令的使用方法。

1. chown命令

chown 命令的作用是更改文件的属主，也可以用于修改文件属组。

【命令格式】

```
chown [option] user:group file
```

【常用选项】

chown 命令的常用选项只有一个 R，该选项的作用是递归地修改目录及目录中的所有文件的属主和属组。

【用法示例】

只有文件的属主或 root 用户可以使用 chown 命令，非 root 用户的属主只能更改文件的属组为用户所在的另一个用户组。即属主只能修改文件的属组，并且修改后的属组必须是属主所在的用户组（私有组、附加组均可）。

（1）更改文件 a 的属主为 user：

```
#使用 chown 命令将文件 a 的属主更改为 user，并使用 ls 命令验证
# chown user a
# ls -l a
-rw-rw-r-- 1 user wlh 9733 Sep 13 15:35 a
```

上面这条命令只能以 root 用户的身份执行。

（2）修改文件 a 的属主为 wlh，属组为 teacher：

```
#以 root 用户身份使用 chown 命令同时修改文件的属主和属组
# chown wlh:teacher a
# ls -l a
-rw-rw-r-- 1 wlh teacher 9733 Sep 13 15:35 a
```

执行完上面的命令后，用户 wlh 将获得文件 a 的所有权。

（3）除 root 用户可以使用 chown 命令外，文件属主也可以使用。例如文件属主 wlh 要把文件 a 的属组更改为其自身所在的另一个组 admin：

```
#文件属主使用 chown 命令修改文件 a 的属主为 admin
$ chown :admin a
$ ls -l a
-rw-rw-r-- 1 wlh admin 9733 Sep 13 15:35 a
```

（4）如果要修改目录及目录中所有文件的属主和属组，可以使用选项 R 递归地处理这些文件。

例如要将目录 test 及其中的所有文件属主修改为 wlh，属组修改为 admin：

```
#使用选项 R 递归地修改目录中所有文件的属主和属组
# chown -R wlh:admin test
# ls -l test
total 24
-rw-r--r-- 1 wlh admin  49 Sep 13 16:41 a
-rw-r--r-- 1 wlh admin   0 Sep 13 16:41 ab
-rw-r--r-- 1 wlh admin   0 Sep 13 16:41 abc
-rw-r--r-- 1 wlh admin 347 Sep 13 16:41 smb.conf
```

注意：普通用户使用 chown 命令时，无法将自己的文件传递给其他用户（即普通用户无法修改文件的属主）。

2. chgrp命令

chgrp 命令用于改变文件的属组。与 chown 命令一样，这个命令也只能由文件属主和 root 用户使用。并且非 root 用户的属主只能将文件的属组修改为用户自身所在的其他用户组。

【命令格式】

```
chgrp [option] group filename
```

【常用选项】

chgrp 命令的常用选项只有一个 R，其作用是递归地修改目录及目录中所有文件的属组。

（1）将文件 a 的属组修改为 teacher：

```
#使用 chgrp 命令修改文件 a 的属组为 teacher，并使用 ls 命令验证
$ chgrp teacher a
$ ls -l a
-rw-rw-r-- 1 wlh teacher 9733 Sep 13 15:35 a
```

（2）chgrp 命令也提供了递归处理文件的选项 R。例如要将目录 test 及其中的所有文件属组都修改为 amin：

```
#使用选项 R 递归地修改 test 目录中的所有文件
$ chgrp -R admin test
```

提示：虽然 Linux 系统为修改属主和属组提供了两个不同的命令，但由于使用 chown 也可以修改属组，因此更多时候都使用 chown 命令。

6.3.3　文件权限及表示方法

与 UNIX 一样，Linux 系统中的传统文件权限有读、写和执行 3 类，这些权限在文件创建时由系统写入。本小节将简单介绍 Linux 系统中的传统文件权限及表示方法。

1. 传统文件权限

Linux 系统中的传统文件权限虽然只有读、写和执行 3 类，但这 3 类权限对于文件和目录而言意义却不相同。

对于文件而言，这 3 类基本权限的含义如下。

- 读：允许读取文件内容。具体形式包括查看文件内容、复制等。
- 写：允许写入内容。具体形式包括编辑文件内容、追加内容、删除文件等。
- 执行：如果该文件是一个可执行的脚本、二进制代码文件或程序，此权限用于控制用户能否执行该文件。

对于目录而言，这 3 类基本权限的含义有所不同。

- 读：允许用户查看目录中的文件列表。例如使用 ls 命令，列出目录中的文件列表。
- 写：允许用户在目录中创建和删除文件。删除目录中的文件时，还应该具备相应文件的写权限。
- 执行：允许用户使用 cd 命令进入目录。

在实际使用时，应该特别注意传统权限对文件和目录的不同含义，以免出现错误。

虽然用户可能无法进入目录或查看目录中的文件列表，但如果目录中的文件权限允许，用户仍然可以使用输入全路径的方法操作文件。

2．权限表示方法之符号模式

在 Linux 系统中，可以使用两种方式表示文件及目录的权限。第 1 种方法称为符号模式，读、写和执行权限分别用字母“r”、“w”和“x”表示。

一般使用 ls 命令的长格式查看文件的权限（配合使用选项 l），此时命令将使用符号模式表示文件的权限。例如查看文件 a 的权限信息：

```
#使用 ls 命令查看文件 a 的详细信息
$ ls -l a
-rw-r--r-- 1 ljx ljx     414 Sep 14 20:20 a
```

上面的输出中，第 1 个字段的第 1 个字符表示文件类型，其后的 9 个字符每 3 个为 1 个权限位。命令输出中的 3 个权限位分别是“rw-”、“r--”和“r--”，这 3 个权限位表示拥有权限的 3 类用户或组。从左起这 3 类用户和组的含义如下。

- 第 1 个权限位：文件的属主权限，用字母 u 表示（可速记为 user）。上面的示例中，文件 a 的属主权限为可读、可写。
- 第 2 个权限位：文件的属组权限，用字母 g 表示（可速记为 group）。上面的文件 a 中，用户组 ljx 中的用户权限为只读。
- 第 3 个权限位：除属主和属组以外的其他用户权限，用字母 o 表示（可速记为 other）。上面的示例中，其他用户的权限为只读。

3．权限表示方法之绝对模式

第 2 种方法使用 8 进制数字表示，通常将这种表示权限的方法称为绝对模式。在绝对模式中各数字对应的权限如下。

- 1：数字 1 表示执行权限。
- 2：数字 2 表示写权限。
- 4：数字 4 表示读权限。

用绝对模式表示多个权限时，直接将这些权限对应的数值相加即可。例如用数字 6 表示可读、写，用数字 7 表示可读、写、可执行等。

绝对模式使用 3 位数字表示不同用户或组的权限，从右起第 1 位表示其他用户权限（即符号模式中的 o），第 2 位表示属组用户权限（即符号模式中的 g），第 3 位表示属主权限（即符号模式 u）。

例如数字 004 表示其他用户可读。要表示属组用户拥有可读、可执行权限，可以使用

数字 050。表示属主可读、写、可执行，可以使用数字 700。将这几个用户和组的权限进行叠加，并用绝对模式完整表达出来应该使用数字 754。

6.3.4　文件权限管理命令 chmod

文件权限管理是 Linux 系统中最基本的管理工作之一，文件权限管理由文件的属主和 root 用户执行。本小节将简单介绍权限管理命令 chmod。

【命令格式】

```
chmod [option] [mode] file
```

【常用选项】

chmod 命令的常用选项只有一个 R，该选项的主要作用是递归地修改目录及目录中所有文件的权限。

【参数说明】

chmod 命令中的 mode 参数是权限表达式，权限表达式可以使用符号模式和绝对模式表示。使用符号模式表示时，表达式可以拆分为操作对象、操作符和权限列表 3 部分。

- 操作对象是用户和组，使用字母 u、g、o 和 a 表示（其中 a 表示 u、g、o，可速记为 all）。如果表达式中没有指定操作对象，chmod 命令将默认使用 a 作为操作对象。
- 操作符表示赋予或收回权限。可以使用的 3 个操作符是："+"、"–" 和 "="，分别表示添加、删除和赋予权限。
- 权限列表：读（r）、写（w）和执行（x）权限的组合。

【用法示例】

（1）修改文件 a 的权限，为属主加上可执行权限：

```
#使用 u+x 表示为属主添加可执行权限
$ chmod u+x a
#使用命令 ls 验证结果
$ ls -l a
-rwxr--r-- 1 ljx ljx 414 Sep 14 20:20 a
```

（2）修改权限时，也可以使用多个操作对象。例如修改文件 a 的权限，去掉属组和其他用户的读权限：

```
#使用 go-r 表示去掉属组和其他用户的读权限
$ chmod go-r a
$ ls -l a
-rwx------ 1 ljx ljx 414 Sep 14 20:20 a
```

（3）也可以对多个对象使用多个权限的组合。例如将文件 a 的权限修改为所有用户均可读写、可执行：

```
#使用表达式 ugo+wrx 表示为所有用户添加可读写、可执行权限
$ chmod ugo+wrx a
$ ls -l a
-rwxrwxrwx 1 ljx ljx 414 Sep 14 20:20 a
```

上面这条命令也可以写成如下形式：

```
#使用表达式 a=wrx 也可以表示为所有用户添加可读写、可执行权限
$ chmod a=wrx a
```

（4）chmod 命令中的权限表达式也可以使用绝对模式表示。例如设置文件 a 的权限为属主可读写、可执行，属组用户可读写，其他用户只读：

```
#使用绝对模式 764 表示属主可读写、可执行，属组用户可读写，其他用户只读
$ chmod 764 a
$ ls -l a
-rwxrw-r-- 1 ljx ljx 414 Sep 14 20:20 a
```

（5）设置文件 a 的权限为属主可读写、可执行，属组及其他用户没有任何权限：

```
#使用绝对模式 700 表示属主可读写、可执行，其他用户没有任何权限
$ chmod 700 a
$ ls -l a
-rwx------ 1 ljx ljx 414 Sep 14 20:20 a
```

（6）要对目录及目录中的所有文件设置权限时，可以使用 chmod 命令的选项 R。例如要将目录 test 及其中所有的文件均设置为所有人可读写：

```
#使用选项 R 递归地修改目录 test 及其中的所有文件的权限
$ chmod -R 666 test
```

使用 chmod 命令修改权限时，符号模式和绝对模式各有优势，读者按需要进行选择即可。

虽然有时可能属主并没有写权限，但是由于其仍然对文件具有管理权限，所以属主依然可以删除这些文件。

6.3.5　suid、sgid 和 sticky 权限概述

在 Linux 系统中还存在一个比较特殊的 set 权限，set 权限可以分为：suid、sgid 和 sticky 权限 3 种。虽然这 3 种权限很少使用，但一些特殊时候使用这 3 种权限却很方便（例如需要普通用户帮 root 用户执行一项任务时）。本小节将简单介绍这 3 种权限的使用方法。

注意：由于 set 权限很容易引发安全问题（一旦出现，往往是致命的安全问题），某些系统可能会出于安全方面的考虑，修改或禁止使用这些权限。感兴趣的读者可以阅读相关文档了解其危险性、具体使用说明等。

1．suid、sgid权限及其使用方法

有时可能希望以某个用户的身份执行一个脚本或命令，例如 root 用户编写了一个用于备份的脚本，并且希望普通用户能执行这个备份脚本。存在的问题是，普通用户可能没有权限读取这些需要备份的文件（与 Windows 系统不同，程序和脚本访问文件时的权限，与执行该程序和脚本的用户访问权限相同），如果因此修改需要备份的文件的访问权限是不明智的。

像上面这种情形，就可以使用 set 权限（由于这 3 个权限名称都以 s 开头，许多时候也称为 s 权限）。包含 s 权限的程序或脚本运行时，其访问文件的权限与程序、脚本的属主或属组的访问权限相同。

【set 权限举例】

一个最典型的例子是普通用户修改密码或默认 Shell 等设置时，需要修改系统用户文件/etc/passwd 和影子文件/etc/shadow。

查看这两个文件的权限：

```
#查看系统用户文件和影子文件的权限设置
$ ls -l /etc/passwd;ls -l /etc/shadow
-rw-r--r-- 1 root root 1677 Sep 14 20:19 /etc/passwd
-r-------- 1 root root 1186 Sep 13 16:03 /etc/shadow
```

从上面的命令输出中可以看到，普通用户均不能修改这两个文件的内容，此时对相关命令使用 s 权限，命令执行时就可以使用 root 用户的身份修改这两个文件了。

使用 ls 命令查看修改用户密码和修改 Shell 的命令文件权限：

```
#查看 passwd 和 chsh 命令文件的权限
# ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 22984 Jan  7  2007 /usr/bin/passwd
# ls -l /usr/bin/chsh
-rws--x--x 1 root root 19096 Jan 20  2010 /usr/bin/chsh
```

在上面的命令输出中，属主权限中的 s 即表示 s 权限。

当用户使用 passwd 和 chsh 命令时，s 权限将允许命令修改系统用户文件/etc/passwd 和用户密码文件/etc/shadow 中的相关设置。

像上面这个例子中，s 权限处于属主权限位时称为 suid，处于属组权限位时称为 sgid。

【设置 s 权限】

s 权限可以设置为以属主的身份运行（suid）程序或脚本，也可以设置为以属组的身份运行（sgid）程序或脚本。

（1）以符号模式设置属主的 s 权限：

```
#使用 u+s 表示为文件添加 suid 权限
# chmod u+s test.sh
# ls -l test.sh
-rwsr-xr-x 1 root root 49 Sep 11 16:32 test.sh
```

（2）以符号模式设置属组的 s 权限：

```
#使用 g+s 表示为文件添加 sgid 权限
# chmod g+s test.sh
# ls -l test.sh
-rwsr-sr-x 1 root root 49 Sep 11 16:32 test.sh
```

（3）在绝对模式中，使用第 4 个权限位（从右算起）表示 s 权限，并用数字 4 表示 suid，数字 2 表示 sgid。

用绝对模式设置 s 权限的同时，还可以设置文件访问权限。例如设置文件 test.sh 的 s 权限时，一并设置其文件访问权限：

```
#将第 4 个权限位设置为 6，表示同时为文件添加 suid 和 sgid 权限
# chmod 6755 test.sh
# ls -l test.sh
-rwsr-sr-x 1 root root 49 Sep 11 16:32 test.sh
```

从上面的 3 个示例命令的输出结果可以看出，在 ls 命令的长格式中，使用字母 s 占用属主和属组的执行权限位的方法，表示文件拥有 s 权限。如果原文件具有可执行权限，就使用小写字母 s 表示有 s 权限和执行权限；如果原文件没有可执行权限，则用大写字母 S 表示具有 s 权限并且没有执行权限。

2. sticky权限及其使用方法

在上一节中讲到，目录的写权限，指的是用户可以在目录中创建或删除文件。然而一些特殊时候，可能希望某个目录中的文件，其他用户能够写入，但不能删除。这时可以给相应的目录添加 sticky 权限，这样其他用户就能使用该目录中具有写入权限的文件，但不能删除这些文件。由于 sticky 权限能阻止用户删除文件，所以经常被称为防删除位。

【sticky 权限应用举例】

在系统临时目录/tmp 中有许多文件，这些文件都是用户及其应用程序使用的临时文件。任何人都能够编辑这个目录中具有写权限的文件，但只有属主才能删除这些文件。这是因为/tmp 目录具有 sticky 权限。

查看系统临时目录/tmp 的权限：

```
#使用 ls 命令查看系统临时目录的权限
# ls -ld /tmp
drwxrwxrwt 7 root root 4096 Aug 14 11:31 /tmp
```

【设置 sticky 权限】

（1）符号模式使用字母 t 表示防删除位。例如为目录 file 添加防删除位：

```
#使用 o+t 表示为目录添加防删除位
# chmod o+t file
# ls -ld file
drwxrwxrwt 2 root root 4096 Sep 18 00:59 file
```

（2）在绝对模式中，在第 4 个权限位（从右算起）使用数字 1 表示防删除位。使用绝对模式添加防删除位的时候，还可以设置目录的权限。例如：

```
#使用第 4 位的数字 1 表示为文件设置防删除位
# chmod 1777 file
# ls -ld file
drwxrwxrwt 2 root root 4096 Sep 18 00:59 file
```

使用命令 ls 的长格式查看权限时，使用字母 t 表示具有防删除位。用小写字母 t 表示原目录有可执行权限和 sticky 权限，字母 T 表示目录没有可执行权限但具有 sticky 权限。

注意：虽然防删除位可以防止用户删除文件，但通常不能阻止 root 和属主删除文件，因为这两类用户具备管理权限。

6.3.6　权限掩码命令 umask

当用户创建文件时，系统会自动为文件添加一个默认权限。如果用户忘记修改文件的默认权限（系统默认的权限可能并不适合用户的所有文件），就可能会引发一些安全隐患。

为避免此类问题，通常建议用户修改创建文件时的默认权限，即将创建文件的默认权限修改为一个相对安全的值。

在 Linux 系统中，使用 umask 值确定用户创建文件的默认权限（由于默认权限使用的管理命令是 umask，因此人们通常称为 umask 值），可以使用修改 umask 值的方法修改用户创建文件时的默认权限。由于 mask 有掩码的意思（类似的还有计算机网络中的网络掩码 netmask），许多时候人们也形象地称其为权限掩码。本小节将简单介绍权限掩码 umask 的使用方法。

权限掩码在用户登录系统时就已经被初始化。umask 值使用 3 位 8 进制数表示默认权限，即绝对模式。对于文件而言，每一位的最高值为 6（默认情况下，系统不允许赋予文件可执行权限，以避免一些可能的安全性问题）。对于目录而言，每一位最高值为 7。

1. 查看umask值

可以直接使用命令 umask 查看当前使用的 umask 值：

```
#使用 umask 命令查看当前使用的 umask 值，并新建文件使用 ls 命令验证
$ umask
0002
$ touch abc
$ ls -l abc
-rw-rw-r-- 1 ljx ljx 0 Sep 16 18:33 abc
```

从上面的命令输出不难看出，umask 值使用的是权限“遮罩”，即使用所有位的最高权限 666，除去权限掩码中被遮罩的权限，得出的结果就是默认权限。

由此可以简单地认为：umsk 值中的权限，是默认情况下不能得到的权限。

新建一个目录并查看其权限：

```
#使用 mkdir 命令新建目录，并查看目录的权限
$ mkdir ab
$ ls -ld ab
drwxrwxr-x 2 ljx ljx 4096 Sep 16 19:00 ab
```

由上面的命令输出可以看出，新建一个目录时，其默认权限是所有位的最高权限 777，去掉 umask 值的结果。

2. 修改权限掩码

要修改用户的 umask 值，直接使用 umask 命令，再加上权限掩码的值即可。例如修改默认 umask 值：

```
#使用 umask 命令修改权限掩码
#022 表示属主可读写，属组和其他用户只读
$ umask 022
```

```
$ umask
0022
#创建新文件和目录，并查看其权限验证效果
$ touch b
$ ls -l b
-rw-r--r-- 1 ljx ljx 0 Sep 16 20:45 b
$ mkdir file
$ ls -ld file
drwxr-xr-x 2 ljx ljx 4096 Sep 16 20:49 file
```

从上面的命令输出可以看出，umask 值一经设置，立即生效。

3．保存修改后的umask值

使用 umask 命令设置的权限掩码将会在系统重启后丢失，如果希望保存 umask 值，通常将其写入用户初始化文件~/.bash_profile 中。

保存权限掩码到用户初始化文件.bash_profile：

```
#将 umask 022 写入用户初始化文件.bash_profile
$ echo "umask 022">>~/.bash_profile
#使用 cat 命令查看用户初始化文件的内容
$ cat ~/.bash_profile
# .bash profile
……
PATH=$PATH:$HOME/bin

export PATH
#以下为新添加的内容
umask 022
```

文件.bash_profile 主要用于初始化用户环境，关于初始化系统环境的过程将在第 16 章中详细介绍。

注意：在理解 umask 值的工作原理时，不应简单地理解为：用最高权限减去 umask 值，即可得到默认权限。举个简单的例子，umask 值中出现权限值 3 时，相减得到的结果将与实际情况不符。

6.4　POSIX ACL 权限系统及其管理命令

在 Windows 系统中，用户可以对文件或文件夹设置多重权限。权限对象可以是一个或多个用户，也可以是一个或多个组，权限的分类也很详细。相比之下，从 UNIX 继承来的权限系统在一些实际应用中，逐渐显现出缺乏灵活性等缺点。例如文件服务器中的权限对象是多用户、多用户组时，传统文件权限系统就无法满足需求。

为此 Linux 系统提供了一个新的遵从 POSIX ACL 标准的权限系统，通常将这个新的权限系统称为 POSIX ACL 权限系统（或 ACL 权限）。本节将简单介绍 POSIX ACL 权限系统及其管理命令。

6.4.1　POSIX ACL 权限系统概述

POSIX 是 Portable Operating System Interface 的缩写，中文含义是可移植操作系统接口最初由 IEEE（电气和电子工程师协会）开发，目的是建立一个统一的标准接口，以便于在不同的系统间移植应用程序。

针对 UNIX 系统权限的不足，POSIX 提出了一个新的权限标准 POSIX ACL。ACL 是 Access Control List 的缩写，中文含义是访问控制列表。用户在访问控制列表中可以定义多用户、多用户组对文件的访问权限。

POSIX ACL 在 Linux 内核的 2.6 版上被正式支持，一些常见的文件系统如 ext2、ext3 和 JFS 等都支持 POSIX ACL。由于 POSIX ACL 不仅需要内核和文件系统的支持，还需要应用程序的支持，因此一些旧版的系统或软件可能不支持 ACL。在实际应用时，应该选用版本较新的发行版或软件，详细信息可以参考相关应用程序的说明文档。

6.4.2　ACL 权限管理和查看命令 setfacl、getfacl

POSIX ACL 仍然使用传统权限中的读、写和执行作为管理的核心，不同的是可以为文件添加多个用户或组的权限细则。为文件添加 ACL 权限可以使用 setfacl 命令。本小节将简单介绍 ACL 权限管理、查看命令 setfacl 和 getfacl。

1. ACL权限管理命令setfacl

setfacl 命令主要用于添加、删除 ACL 权限。与传统文件权限管理的命令一样，只有 root 用户和文件属主才能使用 setfacl 命令。

【命令格式】

```
setfacl [选项] 权限表达式 文件名
```

【常用选项】

- ❑ b：删除所有扩展 ACL 权限。
- ❑ k：删除默认的 ACL 权限。
- ❑ d：设置默认的 ACL 权限。
- ❑ R：递归地设置目录及目录中的所有文件。
- ❑ m：修改、添加已有的 ACL 权限。

【参数说明】

设置 ACL 权限时，还需要使用权限表达式。与 chmod 命令的权限表达式不同，setfacl 命令的权限表达式需要明确地指出用户和组的权限。

ACL 表达式的基本格式如下：

```
[对象类型]:[对象]:权限列表
```

表达式以冒号“:”作为分隔符，将对象类型、对象和权限分为 3 个字段。这 3 个字段的含义及选项如下。

- ❑ 对象类型：对象类型表示要操作的对象类型。可以使用的对象类型有 4 种：u、g、

m 和 o，分别代表用户、组、mask 和其他用户。mask 表示 ACL 权限的安全值。

- 对象：应用权限的对象，用户或组的名称。
- 权限：3 种基本权限（w、r 和 x）的组合。

如果一条命令中要使用多个权限表达式，可以使用逗号“,”将这些权限表达式分隔开。

2．查看ACL权限命令getfacl

设置了 ACL 权限的文件，不能使用 ls 命令查看其附带的 ACL 权限。要查看 ACL 权限，通常使用 getfacl 命令。

【命令格式】

```
getfacl [option] filename
```

使用 getfacl 命令查看文件的 ACL 权限时，通常不需要使用任何选项，直接将要查看的文件作为其参数即可。

注意：文件系统可能默认不支持 ACL，可以通过重新挂载的方式，让这些文件系统支持 ACL，命令为 mount -o remount,acl /dev/sda3。这个命令中的/dev/sda3 为文件系统对应的分区，具体命令及参数的含义将在第 7 章中介绍。

6.4.3　ACL 权限管理

上一小节简单介绍了 ACL 权限管理命令 setfacl，以及查看 ACL 权限命令 getfacl。本小节将通过几个示例讲解如何使用这两个命令管理 ACL 权限。

（1）添加 ACL 权限

以 root 用户新建一个文件 a，查看其权限：

```
#使用 ls 命令查看新建立的文件 a 的权限
# ls -l a
-rw-r--r-- 1 root root 276 Sep 18 21:15 a
```

要给文件添加 ACL 权限，可以配合使用选项 m。例如要给用户 ljx 添加一个可读写权限：

```
#使用表达式 u:ljx:rw 为用户 ljx 添加可读写权限，并使用 ls 命令观察文件权限变化
# setfacl -m u:ljx:rw a
# ls -l a
-rw-rw-r--+ 1 root root 276 Sep 18 21:15 a
```

可以看到文件 a 的权限后面多了一个加号“+”，ls 命令使用这种方式表示这个文件有 ACL 权限。

（2）查看 ACL 权限

要查看文件的 ACL 详细信息，需要使用命令 getfacl。例如查看文件 a 的 ACL 权限信息：

```
#使用 getfacl 命令查看文件 a 的 ACL 权限
# getfacl a
```

```
#getfacl 命令先输出了文件的基本信息
# file: a
# owner: root
# group: root
#当输出的用户名为空时，表示属主和属组的权限
user::rwx
#以下是为用户 ljx 新添加的 ACL 权限
user:ljx:rw-
group::rw-
mask::rw-
other::r--
```

上面的命令输出中，以“#”开头的为注释行，注释行中描述了文件的基本情况，例如当前的文件名为 a，文件的所有者和属组均为 root。紧随其后的是 ACL 权限列表，ACL 权限列表中的条目与 ACL 表达式的格式相同，其中对象为空的字段表示属主、属组和其他用户。

除此之外，ACL 权限列表中还有一个特殊的条目，其对象类型为 mask，其主要作用是限制 ACL 条目的最大权限。

（3）ACL 权限上限 mask

为了安全地使用 ACL 权限，系统为文件添加了一个默认的权限上限，这个权限上限被保存在 mask 条目中。当设置的 ACL 权限超过权限上限时，超过的权限将会被系统屏蔽。

设置 mask 权限上限为只读，添加用户组 admin 的权限为可读、写、可执行，并以 ljx 用户的身份写入新内容作测试：

```
#通过查看系统用户组文件的方法查看用户 ljx 所属组
# cat /etc/group | grep ljx
ljx:x:501:
teacher:x:503:wlh,ljx
admin:x:504:wlh,ljx
teacher1:x:505:wlh,ljx
#使用 setfacl 命令为文件 a 设置 mask 权限上限，并使用 getfacl 命令查看
# setfacl -m m::r,g:admin:rwx a
# getfacl a
# file: a
# owner: root
# group: root
user::rw-
#命令使用注释的方式提示用户 ljx 和组 admin 的权限列表中，只有读权限有效
user:ljx:rw-                  #effective:r--
group::r--
group:admin:rwx               #effective:r--
mask::r--
other::r--
#使用 su 命令切换到 ljx 用户
# su ljx
#使用重定向的方式写入内容
#系统提示用户权限不足
$ echo "hello">>a
bash: a: Permission denied
```

上面这个例子充分说明了 mask 权限上限的作用，即使用户和组有写入权限，由于 mask 的限制，也不能执行写入操作。

mask 可以用来临时调整 ACL 权限，在确认安全的情况下，通常可以将 mask 的值设置为可读写。

（4）删除 ACL 权限

要删除文件的 ACL 权限可以使用 setfacl 的选项 b。例如删除文件 a 的所有 ACL 权限：

```
#使用选项 b 删除文件的所有 ACL 权限
# setfacl -b a
# getfacl a
# file: a
# owner: root
# group: root
user::rw-
group::r--
other::r--
```

删除之后查看文件的 ACL，可以发现命令 getfacl 只输出了文件的传统权限列表。

（5）递归地添加 ACL 权限

有时需要对一个目录及目录中的所有文件设置 ACL 权限，这时可以使用选项 R 递归地为目录中的所有文件添加 ACL 权限。

例如要设置 test 目录中的所有文件的 ACL 权限为 ljx 用户可读写，admin 用户组可读、写：

```
#使用选项 R 递归地设置目录的 ACL 权限
# setfacl -Rm u:ljx:rw,g:admin:rw test
```

（6）预设 ACL 权限

许多时候可能希望为某个目录中新建的文件添加默认的 ACL 权限，即每添加一个文件，系统会自动为文件添加 ACL 权限，这时可以使用选项 b 预设 ACL 权限。

例如要为目录 file 设置预设 ACL：

```
#使用选项 d 为目录设置预设 ACL 权限
# setfacl -dm u:ljx:rw,g:admin:rw file
#新建文件并验证结果
# cd file/
# touch c
# getfacl c
# file: c
# owner: root
# group: root
user::rw-
user:ljx:rw-
group::r-x                   #effective:r--
group:admin:rw-
mask::rw-
other::r--
```

上面的命令为目录设置了预设的 ACL 权限，在目录中建立的新文件将会自动继承预

设的 ACL 权限。

（7）删除预设 ACL 权限

有时可能需要删除一个已经设置好的预设 ACL 权限，这时可以配合使用选项 k。

例如要删除目录 file 的预设 ACL 权限：

```
#使用选项 k 为目录添加预设的 ACL 权限
# setfacl -k file
```

6.5　小　　结

- 6.1 节主要介绍了 Linux 系统中的用户管理，包括系统用户文件、影子文件、添加、删除用户命令、禁止用户登录的方法等内容。
- 6.2 节介绍了 Linux 系统中的用户组管理，包括系统用户组文件、组影子文件、添加删除用户组、将用户加入到组等。用户和用户组管理是多用户系统中不可缺少的一部分，因此初学者应该掌握 6.1 和 6.2 节的内容。
- 6.3 节讲解了 Linux 系统从 UNIX 系统中继承的传统文件权限及管理。权限管理在 Linux 管理中经常用到，因此每个初学者都应该学习如何管理传统文件权限。
- 6.4 节简单介绍了 Linux 系统中的另一个权限系统：POSIX ACL。虽然 ACL 权限在实际应用中较少用到，但初学者应该对其有一定的了解，以便更加灵活地部署应用。

权限管理是 Linux 系统管理中比较重要的部分，本章简单介绍了 Linux 的权限系统及管理方法。其中的 Linux 系统中的用户、用户组、传统文件权限及管理都是重点，每一个初学者都应该掌握。

在部署有关文件权限的应用时（例如 FTP、Samba 服务），通常建议尽量使用服务软件自身的访问控制机制（即用户对文件的读写由服务控制）。

第 7 章　磁盘和文件系统管理

与 Windows 简易的数据存储方式相比，Linux 系统为提高硬盘读写效率，保证数据安全性和可用性，增加了许多较为复杂的技术。因此学习 Linux 系统存储的相关内容时，需要学习 Windows 中从未接触过的许多概念。本章将简单介绍 Linux 系统中最基本的磁盘、文件系统管理，在此基础上，还增加了 RAID（磁盘阵列）、LVM（逻辑卷管理）、磁盘配额、文件系统维护等面向应用级别的知识。

本章主要涉及的知识点如下。

- Linux 系统中的磁盘管理，包括磁盘分区、格式化和添加新磁盘等。
- 文件系统的简单介绍，文件系统挂载、卸载等基本管理。
- RAID 简介、RAID 类别、适用范围，创建和管理阵列等。
- LVM 逻辑卷及应用介绍，创建并管理 LVM 逻辑卷。
- 磁盘配额简介，创建并管理磁盘配额。
- 文件系统维护命令、方法介绍。

小知识：文件系统是操作系统在磁盘等存储介质上存储数据的方法，包括以怎样的组织形式、结构将数据存放在存储介质上等内容。具体地讲就是如何将文件的权限、属主、属组、修改时间、文件大小等信息存放在存储介质上，如何将分区内的文件和目录以快捷有效的方式组织起来，并存放在磁盘上等。这些都是文件系统需要解决的问题。

7.1　磁盘及分区管理

磁盘是目前计算机保存数据的主要介质，无论企业还是个人通常都将数据保存在磁盘上。本节将简单讲解在 Linux 系统中如何查看磁盘设备、为磁盘分区等。

7.1.1　查看磁盘设备列表命令 fdisk

许多时候我们为主机添加了一块新的磁盘，可能是旧的磁盘已经不能满足需求，需要扩容，也可能是更换某个已经损坏的磁盘等。这时就需要为新添加的磁盘分区，创建文件系统。

在执行这些操作之前，应该首先查看新的磁盘是否已经被成功添加。查看系统中的磁盘设备列表，通常使用命令 fdisk。

【命令格式】

```
fdisk [option] device
```

【常用选项】

查看磁盘设备命令 fdisk 只有一个常用选项 l，其作用是列出当前系统中的磁盘设备及分区详情。该命令也可用于磁盘分区，此部分内容将在 7.1.3 节中介绍。

【用法示例】

使用 fdisk 命令查看当前计算机中的磁盘设备列表：

```
#使用选项 l 查看当前计算机中的磁盘设备及分区列表
# fdisk -l
#磁盘 sda 的基本信息
Disk /dev/sda: 85.8 GB, 85899345920 bytes
255 heads, 63 sectors/track, 10443 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

#以下是磁盘 sda 的分区信息
#下面的列表从左至右依次是分区设备、启动标记、起始柱面、结束柱面、块数
#分区类型 Id、分区类型
   Device Boot      Start         End      Blocks      Id   System
/dev/sda1   *           1          50      401593+     83   Linux
/dev/sda2              51        7703      61472722+   83   Linux
/dev/sda3            7704       10313      20964825    83   Linux
/dev/sda4           10314       10443      1044225     5    Extended
/dev/sda5           10314       10443      1044193+    82   Linux swap/ Solaris

#以下是磁盘 sdb 的基本信息
Disk /dev/sdb: 85.8 GB, 85899345920 bytes
255 heads, 63 sectors/track, 10443 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

#由于 sdb 中没有分区，因此以下列表为空
   Device Boot      Start         End       Blocks      Id   System
```

上面的命令输出了系统中使用的所有磁盘，其中 sda 的大小为 85.8GB，255 个磁头，每个磁道有 63 个扇区，一共有 10 443 个柱面。总计有 16 065 个扇区，一个柱面大小为 16065*512（块大小，也称扇区大小）字节。

小知识：在磁盘内部，通常有多张圆形盘片（目前多为 2～3 张），这些盘片的两面都可以存储数据（因此盘片的正反两面都有用于读取、写入数据的磁头）。为了存储数据，人为地以盘片中心为圆心，从外到内划分为多个很小的同心圆，我们将这些同心圆称为磁道。几张盘片中相同位置的磁道形成了一个圆柱体，我们将这个圆柱体称为柱面。为了方便查找数据，将盘片按相同的圆心角划分为许多个扇形区域，我们将这些处于不同磁道的扇形区域称为扇区。读取数据时（写入数据时过程相同），磁盘会先确定需要使用哪个磁头（以此确定数据存放在哪个盘面）。然后确定数据存放在哪个磁道的哪个扇区内。

从上面的命令中可以看出，计算机中除了 sda 外，还有一个磁盘 sdb，并且 sdb 还未分区。

如果使用 fdisk 命令没有列出添加到系统中的磁盘，应该检查磁盘的电源线、数据线是否已经正确接入系统。

提示： 在 Linux 系统中，所有的设备都被放置在目录/dev/中。对于磁盘等存储设备而言，sd 通常表示 SCSI 磁盘、STAT 磁盘、U 盘等设备，hd 通常表示 IDE 磁盘。而软驱通常使用 fd 表示，光驱则使用 hdc 或 cdrom 表示。

7.1.2　查看磁盘设备命令 hdparm

hdparm 命令主要用于查看 IDE 硬盘的工作参数。除此之外，还可以使用此命令设置 IDE 硬盘的工作模式。

【命令格式】

```
hdparm [option] device
```

【常用选项】

- I：显示硬盘提供的硬件信息。
- T：评估硬盘从快速缓存中读取数据的速度。
- t：评估硬盘从缓冲区中读取数据的速度。

该命令还有许多选项，但由于目前大多数用户都已不再使用 IDE 硬盘，因此本书仅讨论 hdparm 命令的部分内容，感兴趣的读者可以阅读相关文档。

【用法示例】

（1）查看硬盘提供的硬件信息：

```
#使用选项 I 查看硬盘 had 提供的硬件信息
# hdparm -I /dev/hda
#以下为硬盘的硬件信息
/dev/hda:
#由于此处使用的是虚拟机硬盘，因此硬件型号、SN 等信息都是虚拟的
ATA device, with non-removable media
        Model Number:       VMware Virtual IDE Hard Drive
        Serial Number:      00000000000000000001
        Firmware Revision:  00000001
Standards:
        Used: ATA/ATAPI-4 T13 1153D revision 17
        Supported: 4 3 2 & some of 5
……
```

注意： hdparm 命令的大多数用法都只能用于 IDE 硬盘。

（2）hdparm 命令最常见的用法是评估硬盘的性能：

```
#使用 hdparm 命令评估硬盘 sda 的性能
# hdparm -tT /dev/sda
```

```
/dev/sda:
#从快速缓存中读取数据的评估结果
 Timing cached reads:   6088 MB in  1.99 seconds = 3064.23 MB/sec
#从缓冲区读取数据的评估结果
 Timing buffered disk reads:   62 MB in  3.11 seconds =  19.93 MB/sec
```

hdparm 命令的这个用法适用于 IDE 硬盘、SCSI 硬盘及 RAID 设备等。

使用 hdparm 命令评估的硬盘性能时，只能反映出硬盘性能的一个方面。其他硬盘性能测试还包括随机存取、顺序存取等，感兴趣的读者可以阅读相关文档了解其他的测试方法。

7.1.3　磁盘分区工具 fdisk

添加新的磁盘后，需要先对磁盘分区，然后才能使用磁盘存储数据。虽然在 Linux 系统中可以使用的分区工具有许多（例如 fdisk、parted 等），但通常都使用 fdisk 工具。

fdisk 是许多发行版都自带的工具，其功能十分强大，且简单易用，非常适合初学者使用。本小节将介绍如何使用 Linux 系统中的磁盘分区工具 fdisk。

1. 分区类型和标识

磁盘分区可以分为主分区、扩展分区和逻辑分区 3 种类型。

- 主分区：可以直接挂载并存取数据。一个磁盘上最多只能有 4 个主分区（即一个磁盘最多只有 4 个可用的主分区）。
- 扩展分区：特殊的主分区（占用一个主分区）。要使用扩展分区，必须先将扩展分区划分为可以直接挂载并存取数据的逻辑分区。
- 逻辑分区：从扩展分区中划分出来的一类分区。逻辑分区可以直接挂载并存取数据，一个扩展分区可以被划分为多个逻辑分区。

从上面的内容可以看出，一个磁盘最多可以划分为 4 个可以直接挂载并存取数据的主分区。如果需要划分为 4 个以上的分区，可以将一部分磁盘空间划分为扩展分区，然后再将扩展分区划分为多个逻辑分区。

以磁盘 sda 为例，sda 的 4 个主分区标识分别为 sda1、sda2、sda3 和 sda4，而逻辑分区则使用 sda5、sda6、sda7……这样的形式表示。

注意：即使主分区不足 4 个，逻辑分区标识也是从 sda5 开始算起的，因此可以使用这种方式辨别分区是否为主分区。

2. 分区工具fdisk

fdisk 命令除了可以查看系统中的磁盘设备之外，还可以对磁盘进行分区。此处以磁盘 /dev/sdb 分区为例进行讲解。

（1）首先使用 fdisk 命令进入分区模式：

```
#以磁盘设备作为参数进入分区模式
# fdisk /dev/sdb
#以下为分区模式的提示信息
```

```
The number of cylinders for this disk is set to 10443.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

#以下为分区模式的命令提示符
Command (m for help):
```

由于磁盘柱面超过 1024，因此 fdisk 给出了警告信息（具体可以参考相关磁盘知识）。

（2）要获取 fdisk 命令的分区模式帮助，可以输入命令 m 并按 Enter 键：

```
#输入帮助命令 m 查看分区模式的帮助信息
Command (m for help): m
#以下为命令列举出的分区模式命令列表
Command action
   a     toggle a bootable flag
   b     edit bsd disklabel
……
   x     extra functionality (experts only)

Command (m for help):
```

fdisk 命令的帮助列表和说明都非常详细，列表中常用的命令及其含义如下。

- d：删除分区。使用此命令时，磁盘上必须有可供删除的磁盘分区。
- l：查看分区类型列表。
- n：添加新分区。使用此命令时，磁盘上必须有空闲的磁盘空间或者还有未被分配的扩展分区空间。
- p：打印当前分区表。由于可能执行过分区操作，因此该命令的输出可能与真实情况不符。
- q：不保存已更改的内容并退出。使用这个命令后，所有在 fdisk 命令中执行过的分区操作都不会被保存到磁盘上。
- t：修改分区类型。
- v：验证分区表。
- w：保存所做的更改并退出。使用此命令后，在 fdisk 命令中执行的分区操作将会同步到磁盘中。此操作可能会丢失磁盘中的所有数据，因此如果磁盘上还有未备份的数据，需要谨慎使用该命令。

使用 fdisk 命令分区时，fdisk 先将磁盘内的分区表读取到缓存中。用户修改分区表时，fdisk 只修改缓存中的分区表，除非执行写入操作（即 w 命令），否则不会影响到磁盘中的分区表。

7.1.4　利用 fdisk 工具对磁盘分区

上一小节介绍了磁盘分区工具 fdisk 及其交互模式中的命令等内容。本小节将通过示

例讲解如何使用 fdisk 工具对磁盘进行分区。

（1）现在对磁盘 sdb 进行分区，将其划分为两个主分区，第 1 个是 sdb1，大小为 40GB。第 2 个主分区为使用全部剩余空间的扩展分区。最后将扩展分区的所有空间划分为逻辑分区 sdb5。

提示： 如果要分区的磁盘中已经存在分区，可以使用命令 d 将已存在的分区删除。

实现上述要求的操作步骤如下：

```
#首先使用命令 p 查看磁盘中是否已存在分区
Command (m for help): p

Disk /dev/sdb: 85.8 GB, 85899345920 bytes
255 heads, 63 sectors/track, 10443 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

#分区列表为空，表示磁盘中没有分区
   Device Boot      Start         End      Blocks   Id  System

#使用命令 n 建立第 1 个主分区
Command (m for help): n
#命令要求用户输入第 1 个分区的类型
#此时输入命令 e 表示扩展分区，命令 p 表示主分区
#本例中输入 p 建立第 1 个主分区
Command action
   e   extended
   p   primary partition (1-4)
p
#命令要求用户输入分区号
#本例中将建立 sdb1，因此输入数字 1
Partition number (1-4): 1
#要求用户输入起始柱面号
#本例中按 Enter 键使用默认的起始柱面 1
First cylinder (1-10443, default 1):
Using default value 1
#要求用户输入结束柱面号或分区大小，默认值为磁盘的最后一个柱面号（即全部空间）
#本例中使用+40G 表示分区大小为 40GB
Last cylinder or +size or +sizeM or +sizeK (1-10443, default 10443): +40G

#使用命令 n 和 e 建立扩展分区
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
e
#扩展分区为 sdb2，因此本例中使用分区号 2
Partition number (1-4): 2
#起始柱面和结束柱面均使用默认值，表示扩展分区使用全部未分配空间
First cylinder (4865-10443, default 4865):
Using default value 4865
Last cylinder or +size or +sizeM or +sizeK (4865-10443, default 10443):
```

```
Using default value 10443

#使用命令 n 和 l 建立逻辑分区
Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
#起始柱面和结束柱面均使用默认值，表示逻辑分区使用全部扩展分区空间
First cylinder (4865-10443, default 4865):
Using default value 4865
Last cylinder or +size or +sizeM or +sizeK (4865-10443, default 10443):
Using default value 10443
```

在上面这个过程中，先使用命令 p 查看当前磁盘中的分区表，然后使用命令 n 依次建立主分区 sdb1，扩展分区 sdb2 和逻辑分区 sdb5。

在 fdisk 命令的交互中，输入分区大小时可以输入柱面号，也可以输入分区大小。本例中使用+40G 表示分区大小为 40GB，除此之外，也可以使用+40000M、+1000000K 等其他形式。如果不输入任何单位，fdisk 命令将认为用户输入的是柱面号。

（2）分区完成后，应该先检查并验证无误后再保存：

```
#使用命令 p 输出缓冲区中保存的分区表
Command (m for help): p

Disk /dev/sdb: 85.8 GB, 85899345920 bytes
255 heads, 63 sectors/track, 10443 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

#以下为缓冲区中的分区列表
   Device Boot      Start         End      Blocks      Id   System
/dev/sdb1               1        4864    39070048+     83   Linux
/dev/sdb2            4865       10443    44813317+     5    Extended
/dev/sdb5            4865       10443     44813286     83   Linux

#使用命令 w 将当前缓冲区中的分区表同步到磁盘中
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

上面这个示例中，先使用命令 p 输出并检查当前缓冲区中的分区列表是否正确，然后使用命令 w 将分区表同步到磁盘中。

如果不需要将缓冲区中的分区列表同步到磁盘，可以使用命令 q 退出，并不保存已执行的操作。

（3）执行完以上操作后，分区工作就已经完成了。但有时内核会无法识别新添加的分区，这个问题主要存在于一些移动硬盘、U 盘等移动存储设备及较早的发行版中。主要现象为设备目录/dev 中，没有相应的分区文件或分区文件不完全等。这时可以使用命令 partprobe，让内核重新读取分区表。

使用 partprobe 命令时，可以不加任何参数和选项：

```
#使用 partprobe 重新读取分区表
# partprobe
```

该命令没有任何返回信息，用户可以通过查看/dev 目录中分区文件的方式验证。

7.2　文件系统管理

“文件系统类型”这个词，相信许多初学者都不陌生。安装 Windows XP 时，安装程序会询问用户使用FAT32还是NTFS文件系统类型格式化分区。许多用户都会选择使用NTFS格式化分区，但二者之间的区别大多数人却不清楚。

通过查阅资料可以了解到，FAT32 文件系统的单个分区容量理论上可以达到 8TB（1TB=1024GB）。而在 Windows 2000 和 XP 中，单个分区实际只能达到 32GB，单个文件最大为 4GB。NTFS 分区则支持最大分区为 2TB，单个文件最大 2TB。除此之外，NTFS 还具有权限、日志文件等特性，这些特性都是文件系统类型决定的。优秀的文件系统不仅读写速度快，而且在数据完整性和功能等方面都具有很大的优势。

与 Windows 系统不同，Linux 系统可以使用许多文件系统。每种文件系统都有独特的优势，在使用这些文件系统之前，应该对其有所了解。本节将简单介绍 Linux 系统中的文件系统管理。

7.2.1　Linux 系统支持的文件系统

作为一个服务器操作系统，Linux 支持许多优秀的文件系统。不同的文件系统在各个方面拥有不同的优势，因此应该了解文件系统的特性，以便部署应用时能选择合适的文件系统。本小节将简单介绍 Linux 操作系统中常见的文件系统及其特性。

Linux 系统支持的文件系统类型及其特性如下。

- ext2：ext2（The Second Extended File System）是早期 Linux 系统使用的文件系统，其读写磁盘的性能十分优异。
- ext3：ext3 是对 ext2 文件系统的扩展，目前许多发行版都默认使用此文件系统。ext3 在 ext2 的基础上添加了日志功能（写入数据时，先将要执行的操作写入日志中，然后再执行实际的操作），该功能可以最大限度地保证数据的完整性。
- ext4：ext3 的后继版本。支持最大单个分区为 1EB（1EB=1024TB=1024*1024GB），最大单个文件为 16TB。虽然其稳定版本发布至今已近两年，但 ext4 还存在一些小缺陷而没有被大范围应用。
- ReiserFS：因其作者是 Hans Reiser 而得名。ReiserFS 在处理大文件和众多小文件时效率很高，经常用在代理服务器、邮件服务器系统中。
- XFS：XFS 是 SGI（Silicon Graphics, Inc.，硅谷图形公司）于 20 世纪 90 年代发布的。这是一个非常优秀的文件系统，功能十分丰富且具备可伸缩性，能支持最大 18EB 单分区及最大 9EB 的单个文件。

- ❑ JFS：JFS 是由 IBM 开发的一种字节级日志文件系统，是一个企业级的文件系统。JFS 采用了数据库技术，因此 JFS 可以在很短的时间内将磁盘数据恢复一致。
- ❑ NFS：NFS（Network File System）是一个基于网络的文件系统，用户可以像使用本地磁盘分区那样使用远端的磁盘空间。目前主要用于远程网络存储等方面。
- ❑ iso9660：标准 CD-ROM 文件系统，要求数据只能以连续的方式存放在 CD 上。

除此之外 Linux 也支持另一些文件系统，例如 vfat、hpfs 和 sysv 等。感兴趣的读者可以阅读相关文档，此处不再一一赘述。

Linux 系统支持的文件系统各有优缺点，因此在部署大型应用时，应该按需要选择文件系统的类型。

虽然 Linux 系统能使用许多文件系统，但对于大多数普通用户而言，使用操作系统默认的文件系统即可满足需求（大多数发行版使用 ext3 作为默认的文件系统，少数最新的系统使用 ext4 作为默认的文件系统）。

提示：Linux 系统使用了磁盘碎片避免机制，因此不需要磁盘碎片整理。关于碎片避免机制，读者可以阅读相关文档。

7.2.2　创建文件系统命令 mkfs

分区在使用前，应该对其进行初始化（主要目的是将数据组织结构等写入分区），这个过程在 Windows 系统中叫做格式化，在 Linux 系统中称为创建文件系统。本小节将简单介绍如何使用 mkfs 命令创建文件系统。

在 Linux 系统中，创建文件系统可以使用 mkfs 系列命令。按创建的文件系统不同，mkfs 系列命令包括：mkfs.ext3、mkfs.ext2 和 ext.vfat 等。利用这些命令可以为分区创建 ext3、ext2 和 fat32 等不同的文件系统，也可以直接使用 mkfs 命令创建不同的文件系统。

如果要查看 mkfs 系统命令，可以在命令模式中输入 mkfs.，然后连按两次 Tab，命令补全功能将列出所有支持的系列命令。除 mkfs 之外，Linux 系统中还存在 mount、fsck 等系列命令。

【命令格式】

```
mkfs [option] device
```

【常用选项】

- ❑ t：用于指定创建的文件系统类型。
- ❑ L：创建文件系统的同时，为文件系统添加卷标。
- ❑ c：创建文件系统前，先检查指定分区的坏道。

【用法示例】

创建文件系统时，必须要向 mkfs 命令指明要创建文件系统的分区和文件系统类型。创建文件系统的分区可以是磁盘分区、U 盘等存储设备，这些设备以文件的形式放在目录 /dev 中。

为磁盘分区创建文件系统会丢失分区中的所有数据，因此创建文件系统之前应该备份分区中的数据。

（1）例如，为/dev/sdb1 分区创建 ext3 文件系统：

```
#使用选项 t 指定创建的文件系统类型为 ext3
# mkfs -t ext3 /dev/sdb1
mke2fs 1.39 (29-May-2006)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
10485760 inodes, 20970841 blocks
……
This filesystem will be automatically checked every 24 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

命令在创建文件系统时，还显示了分区的相关信息，包括块大小、超级块等内容。

（2）也可以使用 mkfs 系列命令创建文件系统。例如：

```
#用 mkfs.ext3 命令创建文件系统的同时，使用选项 c 检查磁盘中的坏道
# mkfs.ext3 -c /dev/sdb1
```

在这个示例中，除使用了选项 c（通常只有老旧的磁盘才使用选项 c 检查坏道）以外，其他都与上一条命令的效果完全相同。

（3）在为分区创建文件系统的同时，有可能还需要为其创建一个卷标，这时可以配合使用选项 L。

例如创建文件系统的同时，为其指定一个卷标/data：

```
#创建文件系统的同时，使用选项 L 为 sdb1 创建卷标
# mkfs -t ext3 -L /data /dev/sdb1
```

本书仅以 ext3 为例，介绍如何创建文件系统。如果用户需要创建操作系统中自带的文件系统（自带的文件系统一般有 msdos、ext2 等），可以参照以上命令执行，此处不再一一赘述。

如果用户需要创建操作系统中没有安装的文件系统（例如 JFS），可以参考文件系统的相关说明，安装文件系统支持，然后再创建文件系统。

7.2.3　查看和修改卷标命令 e2label

卷标是用户对文件系统添加的别名，通常用于标识文件系统的用途。在 Linux 系统中，卷标也可以用于挂载文件系统。本小节将简单介绍查看和修改卷标命令 e2label。

【命令格式】

```
e2label device [newlabel]
```

【参数说明】

使用 e2label 命令时，必须要为其指定一个文件系统。如果用户使用了 newlabel 参数（新的卷标），e2label 命令将修改文件系统的卷标，否则就输出文件系统的卷标。

【用法示例】

（1）对于已经存在文件系统的分区，可以使用 e2label 命令修改卷标。例如将文件系统

/dev/sdb1 的卷标修改为/file：

```
#使用 e2label 命令修改 sdb1 的卷标
# e2label /dev/sdb1 /file
```

命令执行成功后将不会返回任何结果。

（2）e2label 除了可以为文件系统添加、修改卷标外，还可以查看文件系统的卷标。例如查看文件系统/dev/sdb1 的卷标：

```
#不使用 newlabel 参数，查看卷标
# e2label /dev/sdb1
/file
```

7.2.4　挂载文件系统

在 Linux 等类 UNIX 系统中，创建文件系统后，必须先将文件系统挂载到某个目录，然后才能使用该文件系统存储数据。挂载是将文件系统的分区“挂”在某个目录下，这个目录就是该分区的访问点。

举例说明：将分区/dev/sdb1 挂载到目录/mnt/sdb1，挂载后向目录/mnt/sdb1 内写入的数据实际将保存到分区/dev/sdb1 内。此时将目录/mnt/sdb1 称为/dev/sdb1 的挂载点。

使用挂载方式访问磁盘分区的好处非常明显，可以将系统中的任意目录挂载到不同的分区中，如此就可以分割不同类型的数据。本小节将简单介绍如何挂载文件系统。

提示： 可能读者会觉得 Linux 系统中使用分区的过程较为麻烦，但与 Windows 相比，Linux 系统中使用挂载方式访问硬盘分区更具灵活性。管理员可以随时使用新分区，自由扩充某个目录。

在 Linux 系统中，挂载分区使用命令 mount。mount 命令不仅可以挂载分区，还可以挂载光驱、U 盘、光盘镜像等设备和文件。

【命令格式】

```
mount [option] device directory
```

在 mount 命令的参数中，device 表示要挂载的设备或文件，directory 表示设备或文件的挂载点。

由于 mount 命令不能创建挂载点目录，因此在挂载之前，还需要使用 mkdir 命令创建挂载点目录。

【常用选项】

- ❑ a：挂载所有可以挂载的文件系统，如果不指定，则挂载文件/etc/fstab 中列出的文件系统（文件/etc/fstab 用于保存需要自动挂载的文件系统，将在 7.2.6 小节中介绍）。
- ❑ o：指定挂载时使用的参数。
- ❑ t：指定文件系统的类型。如果挂载常见的文件系统，一般不必指定文件系统的类型。通常只有光盘镜像、远程文件系统等需要指定文件系统类型。

在上面的选项中，选项 o 指定的参数通常用于设定挂载细节，常用的参数及功能如表 7.1 所示。

表 7.1　常用的参数

参数	参 数 功 能
async	以非同步的方式读写文件系统
atime	每次存取都更新文件的访问时间戳记
auto	自动挂载。该选项只能用于/etc/fstab 文件中，表示用户使用选项 a 时，此文件系统将被自动挂载
user	允许普通用户挂载文件系统，此选项还附带有 noexec、nodev 等选项（除非挂载时重新指定这些参数，否则默认不生效）
users	允许所有用户挂载文件系统，此选项附带有 noexec、nodev 等选项（除非挂载时重新指定）
dev	挂载后可读取文件系统上的字符设备和块设备文件
exec	挂载后可执行文件系统上的二进制文件
noatime	与 atime 参数作用相反，不更新访问时间戳记。此参数通常用于挂载光盘等只读文件系统
noauto	与 auto 参数作用相反，表示不要自动挂载的文件系统。该参数通常用于光盘驱动器
nouser	禁止普通用户（非 root 用户）挂载文件系统
noexec	与 exec 选项相反，不允许执行文件系统上的二进制文件
nodev	与 dev 选项相反，不允许读取文件系统上的字符设备和块设备文件
remount	试图重新挂载一个文件系统，通常用于改变文件系统的挂载参数
ro	以只读方式挂载文件系统。通常用于修复文件系统或其他不允许写入的情况
rw	以可读写方式挂载文件系统
sync	所有对文件系统的读写操作都应该立即完成（即同步读写文件系统）
suid	设置用户 ID 和组 ID
loop	将文件当成文件系统挂载
nosuid	与 suid 功能相反，不设置用户 ID 和组 ID
bind	将一个子树挂载到其他地方，即从多个位置访问同一文件系统
defaults	默认参数，包括：rw，suid，dev，exec，auto，nouser 和 async 7 个参数

除以上参数外，选项 o 还有一些不常用的参数，读者可以参考相关文档，此处不再一一赘述。

虽然 mount 命令的选项 t 可以指定多种文件系统的类型，但仅局限于已安装的文件系统类型。因此如果要挂载未安装的文件系统，需要事先安装相关文件系统的支持软件包。

注意：如非必要，不要使用 user 和 users 等参数，否则可能会引起系统安全性问题。

【用法示例】

（1）虽然 mount 命令的选项 o 提供了许多让人无从选择的参数，但大多数情况下，只需要使用 defaults 参数即可满足需求。也可以不使用选项 o，让 mount 命令自动选择 defaults 参数。

例如将/dev/sdb1 挂载到目录/mnt/sdb1：

```
#挂载时使用选项 t 指定文件系统的类型，并通过查看挂载点的方式验证
#由于没有指定挂载参数，mount 命令将会自动使用 defaults 参数
# mount -t ext3 /dev/sdb1 /mnt/sdb1
# cd /mnt/sdb1/
```

```
# ls
lost+found
```

上面的命令输出了一个目录 lost+found，这个目录只存在于文件系统的根目录中，因此可以使用此方法判断文件系统是否已经挂载成功。

（2）使用 mount 命令挂载文件系统时，也可以不使用选项 t 指定文件系统的类型：

```
#挂载文件系统时，省略选项 t，让 mount 自动判断文件系统并挂载
# mount /dev/sdb1 /mnt/sdb1
```

只有挂载常见的文件系统时，才可以省略选项 t。

（3）如果不指定 mount 命令的选项和参数，命令会显示系统中已经挂载的所有文件系统：

```
#使用 mount 命令查看系统中已挂载的文件系统
# mount
#以下为当前系统中挂载的文件系统
/dev/sda2 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
……
/dev/sdb1 on /mnt/sdb1 type ext3 (rw)
```

可以使用此方法查看系统中已挂载的文件系统的详细信息。

（4）虽然目前不少系统都为光驱添加了自动挂载功能，但有时仍然需要手动挂载光驱。光驱在 Linux 系统中的标识通常为/dev/cdrom 和/dev/hdc。

将光驱挂载到目录/media：

```
#由于光驱为只读设备，因此 mount 命令提示用户使用只读的方式挂载
# mount /dev/cdrom /media
mount: block device /dev/cdrom is write-protected, mounting read-only
#由于/dev/hdc 也表示光驱，因此也可以通过挂载/dev/hdc 的方式挂载光驱
# mount /dev/hdc /media
```

（5）有时需要使用光盘镜像，这时可以像挂载文件系统那样，直接挂载一个光盘镜像文件：

```
#使用 loop 参数挂载光盘镜像文件
# mount -o loop linux.iso /media
```

上面的命令中，参数 loop 表示将文件当作文件系统挂载到系统中。

（6）许多时候，使用 U 盘、移动磁盘等移动存储设备，临时存放数据。在 Linux 系统中使用这些移动存储设备之前，也需要先挂载移动存储的文件系统。U 盘等移动存储设备通常都挂载到目录/media 或/mnt 中。

提示：由于 U 盘等移动设备使用的文件系统可能不是较为常见的文件系统（例如 Windows 系统的 NTFS 文件系统），因此挂载时可能还需要指定文件系统类型或安装相关软件才能使用。

例如将 U 盘挂载到/media：

```
#使用 mount 命令挂载 U 盘文件系统
```

```
# mount /dev/sde1 /media
```

如果无法确定 U 盘等移动设备的标识符，可以使用命令 fdisk 的选项 l 列出所有的设备，然后通过容量查找对应的标识符：

```
#使用 fdisk 命令查找 U 盘对应的标识符
# fdisk -l
Disk /dev/sda: 85.8 GB, 85899345920 bytes
255 heads, 63 sectors/track, 10443 cylinders
……
#以下是一个容量为 4GB 的 U 盘设备
Disk /dev/sde: 4009 MB, 4009754624 bytes
124 heads, 62 sectors/track, 1018 cylinders
Units = cylinders of 7688 * 512 = 3936256 bytes

   Device Boot      Start         End      Blocks    Id     System
/dev/sde1              1         1018     3913161    83     Linux
```

从上面命令的输出中可以看出，通过查看磁盘的大小就可以确定 U 盘的设备标识符。

除了使用 fdisk 命令查看标识符外，当 U 盘接入系统时，可能还会出现以下提示：

```
#插入 U 盘时，系统会出现以下提示
#以下信息显示了插入 U 盘的品牌、型号、标识符和容量等信息
Vendor: Lenovo    Model: USB Flash Drive    Rev: 1100
  Type:   Direct-Access                     ANSI SCSI revision: 00
SCSI device sde: 7831552 512-byte hdwr sectors (4010 MB)
sde: Write Protect is off
sde: assuming drive cache: write through
SCSI device sde: 7831552 512-byte hdwr sectors (4010 MB)
sde: Write Protect is off
sde: assuming drive cache: write through
sd 3:0:0:0: Attached scsi removable disk sde
sd 3:0:0:0: Attached scsi generic sg4 type 0
```

从上面的提示信息中，也可以看出 U 盘的标识符。

提示： 如果想了解选项 o 的参数用法，可以参阅官方文档，自己动手安装一个 Gentoo（Gentoo 是一个完全由用户自定义组件的 Linux 操作系统）。

7.2.5　卸载文件系统命令 umount

使用完 U 盘、光盘、移动硬盘等移动存储设备后，需要先卸载已经挂载的文件系统，然后再断开与主机的连接。在 Linux 系统中，卸载文件系统可以使用命令 umount。

【命令格式】

```
umount [option] directory
```

使用 umount 命令卸载文件系统时，通常不需要使用任何选项。

【用法示例】

直接将挂载点作为 umount 命令的参数即可卸载。例如将挂载到目录/media 的文件系

统卸载：

```
#使用 umount 命令将挂载到/media 目录的文件系统卸载
# umount /media
```

卸载文件系统之前，应该先确认是否有用户或程序正在使用该文件系统。无论是对挂载文件系统执行读操作还是写操作，都会导致卸载失败。

如果用户没有将已挂载的文件系统卸载，计算机关机时，系统会自动卸载所有的文件系统。

7.2.6　利用 fstab 自动挂载文件系统

如果每次使用磁盘分区都要手动挂载、卸载，将会是一件很繁琐的事情。此时可以使用 fstab 文件自动挂载文件系统。Linux 系统启动时会自动挂载文件/etc/fstab 中记录的文件系统，因此可以利用这个文件自动挂载经常使用的磁盘分区。

1．fstab文件及其格式

系统自动挂载文件/etc/fstab 用于系统启动时自动挂载根文件系统，查看其内容如下：

```
#使用 cat 命令查看/etc/fstab 的内容
# cat /etc/fstab
LABEL=/             /             ext3    defaults        1 1
LABEL=/home         /home         ext3    defaults        1 2
LABEL=/boot         /boot         ext3    defaults        1 2
tmpfs               /dev/shm      tmpfs   defaults         0 0
devpts              /dev/pts      devpts  gid=5,mode=620   0 0
sysfs               /sys          sysfs   defaults         0 0
proc                /proc         proc    defaults         0 0
LABEL=SWAP-sda5     swap          swap    defaults         0 0
```

这个文件以行为单位列出了当前系统中需要挂载的文件系统，每行使用空格分隔为 6 个字段。从左起这 6 个字段及其含义如下（以文件的第 1 行为例说明）。

- 第 1 个字段为要挂载的文件系统。本例中要挂载的文件系统是卷标为/的设备。
- 第 2 个字段是挂载点。本例中的挂载点为/（根目录）。
- 第 3 个字段是文件系统格式。本例中的文件系统格式为 ext3。
- 挂载的参数。如果有多个就用逗号“,”隔开。本例中的挂载参数为 defaults。
- 是否需要备份。0 表示不备份，1 表示要备份。一般根分区设置为 1，其他分区设置为 0。由于本例为根分区，因此设置为 1。
- 指定自检的顺序。一般根分区为 1，其他分区为 2，0 表示不检查。

在上面这个配置文件中，使用了“LABEL=”这样的形式表示使用卷标确认要挂载的文件系统。这样做的好处是，即使磁盘分区标识符发生了变化（更改磁盘接口或添加新硬盘均有可能导致磁盘标识符发生变化），也可以通过卷标自动挂载文件系统，而不至于产生错误。

2．设置自动挂载

了解自动挂载文件中各字段的含义后，可以按上面的格式修改配置文件，并实现自动

挂载文件系统。

例如要设置自动挂载/dev/sdb1，可以在文件结尾添加以下内容：

```
#使用设备自动挂载文件系统
/dev/sdb1            /mnt/sdb1              ext3    defaults          0  0
#使用卷标自动挂载文件系统
LABEL=/data          /mnt/sdb1              ext3    defaults          0  0
```

上面两行分别使用了两种不同的方式挂载文件系统，第 1 种是直接使用设备名的方式，第 2 种为使用卷标的方式。这两种挂载文件系统的方式各有优点，读者按需要选择即可。

添加以上内容后，系统启动时就会自动将/dev/sdb1 挂载到/mnt/sdb1。

除此之外，设置自动挂载时，还需要注意以下几点。

- 如果要挂载的目录中已经存在数据，应该先将目录中的数据拷贝到分区中，否则挂载之后将无法看到该目录中的原有内容。
- 一般不要将光驱、U 盘等移动设备添加到/etc/fstab 文件中，这些设备更适合手动挂载。

注意： 如果需要关机并从计算机上取出磁盘，应该先删除/etc/fstab 中的自动挂载设置，然后再关机并取出磁盘设备，以免发生错误无法启动系统。

7.3　RAID 设备

RAID 是 Redundant Array of Independent Disk 的缩写，中文含义为独立冗余磁盘阵列（通常简称为磁盘阵列），1987 年诞生于加州大学伯克利分校。磁盘阵列的实质是将多个磁盘通过 RAID 控制器组合在一起，形成一个新的磁盘。这个新的磁盘具有更大的容量（将多个磁盘合并使用）、更好的容错性（一份数据多个拷贝），以及更快的读写速度（多磁盘同时读写）等优势。

RAID 技术对数据存储产生了巨大的影响。时至今日，这一技术仍然广泛地应用于许多企业的服务器中。由于 RAID 技术使用范围非常广，目前它已经成为系统管理员的一项重要知识。本节将简单介绍 Linux 系统中软件 RAID 及其使用方法。

7.3.1　磁盘阵列的种类

磁盘阵列将多个廉价、容量小的磁盘组合在一起，使其具备容量大、读写速度快、容错性较好等特点。对于计算机而言，磁盘阵列就是一个磁盘，可以在磁盘阵列上创建文件系统，自由地挂载并存储数据。

磁盘阵列按实现方式可以分为硬件 RAID 和软件 RAID，其区别如下。

- 硬件 RAID：常见的硬件 RAID 以阵列卡等形式存在。对于主机而言，一个硬件 RAID 系统就是一个磁盘。
- 软件 RAID：软件 RAID 通过底层代码实现，通常是由操作系统提供的相关功能。

硬件 RAID 与软件 RAID 各有优势，二者之间的对比如表 7.2 所示。

表 7.2　硬件RAID与软件RAID对比

磁盘类型	安装驱动程序	使用系统资源	支持的存储设备种类	花费资金
软件 RAID	由操作系统内置，无须安装驱动程序	需要使用系统计算资源	可以支持更多的磁盘设备	由操作系统内置，无须购买额外的设备
硬件 RAID	需要为阵列卡安装驱动程序	硬件 RAID 卡包含处理器，无须使用系统资源	仅支持有限的磁盘设备	需要额外购买阵列卡

从上面的对比可以看出，硬件 RAID 虽然安装较为麻烦，但可以不必占用系统资源，从而提高了系统资源利用率，因此硬件 RAID 一般用在专用存储或磁盘读写压力较大的服务器上。软件 RAID 部署比较方便，且能够支持更多的存储设备，因此软件 RAID 一般用在小型服务器、个人和家庭等存储领域。

7.3.2　磁盘阵列级别

磁盘阵列按存储数据的方式不同，可以使用 RAID 0～RAID 7 来表示，通常将其称为阵列级别。除了 0～7 这 8 个级别外，一些厂商还在原有的 8 个级别的基础上，扩展出其他的级别，例如 RAID50、RAID10 等。

阵列级别不同，其存储数据的方式、附带的功能、特性也有所差异。部署磁盘阵列之前应该了解阵列级别及其特性，以便按需要选择阵列级别。本小节将简单介绍几个最常用的阵列级别。

1．RAID 0

RAID 0 的工作原理如图 7.1 所示。

RAID 0 必须由两块以上磁盘组成，存储数据时 RAID 0 会先分割要存入的数据，然后再交错地存储在组成阵列的所有磁盘上。RAID 0 交错地存储数据带来的特性如下。

- 大大提高了读写数据的速度（由于系统总线等多方面因素限制，其读写数据的速度往往低于所有磁盘读写速度的总和）。
- RAID 0 阵列的可用空间是所有磁盘空间的总和。
- 由于数据都没有备份，因此 RAID 0 不提供数据冗余功能。

由于阵列中的数据分布在所有磁盘上，因此有时人们也将 RAID 0 称为带区卷。

注意：无论阵列的级别是多少，组成阵列的磁盘大小都应该大致相等。

2．RAID 1

RAID 1 必须由两块以上磁盘组成，其工作原理如图 7.2 所示。

存储数据时，RAID 1 将要存储的数据完整地存放在组成阵列的所有磁盘上，即每个磁盘都拥有一个拷贝。RAID 1 的特性如下。

- 由于每个磁盘都保存着一份相同的数据，因此 RAID 1 能够提供数据冗余功能。

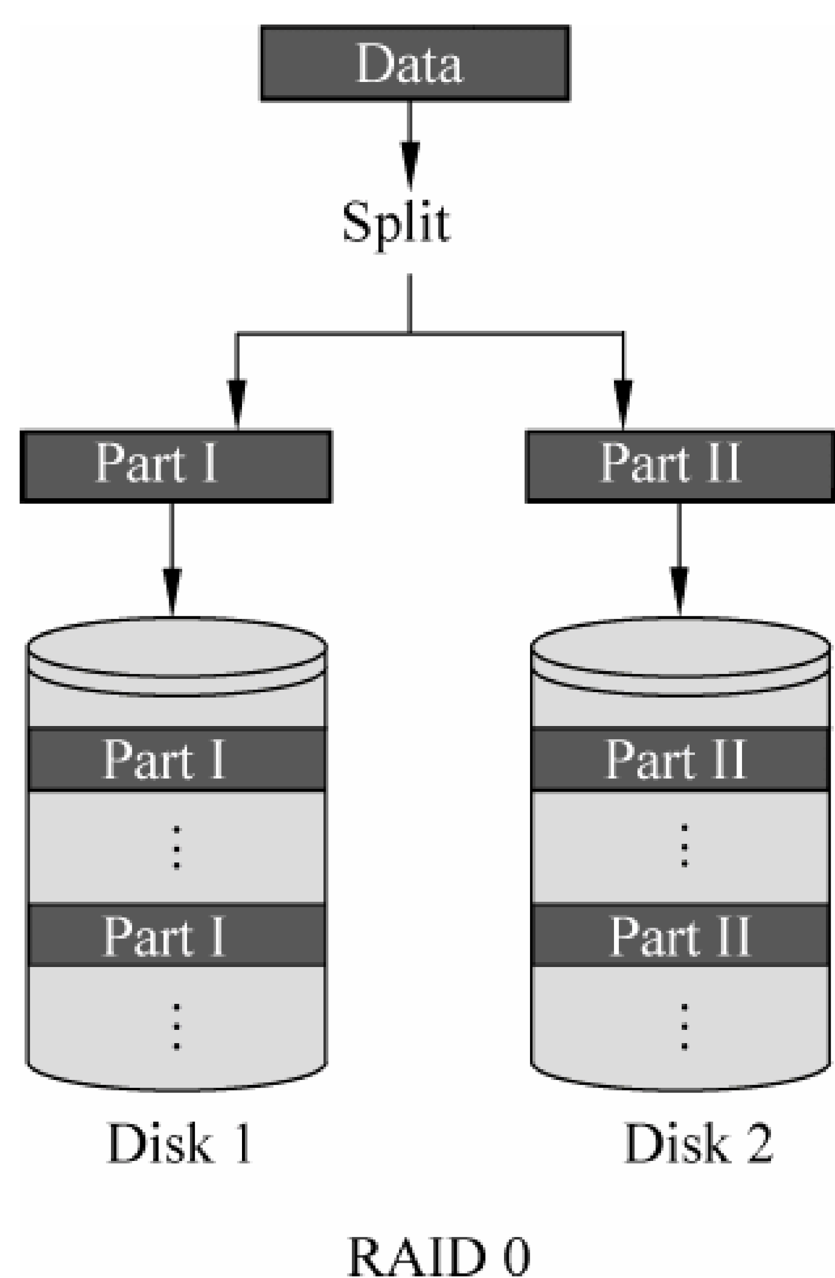


图 7.1　RAID 0 工作原理

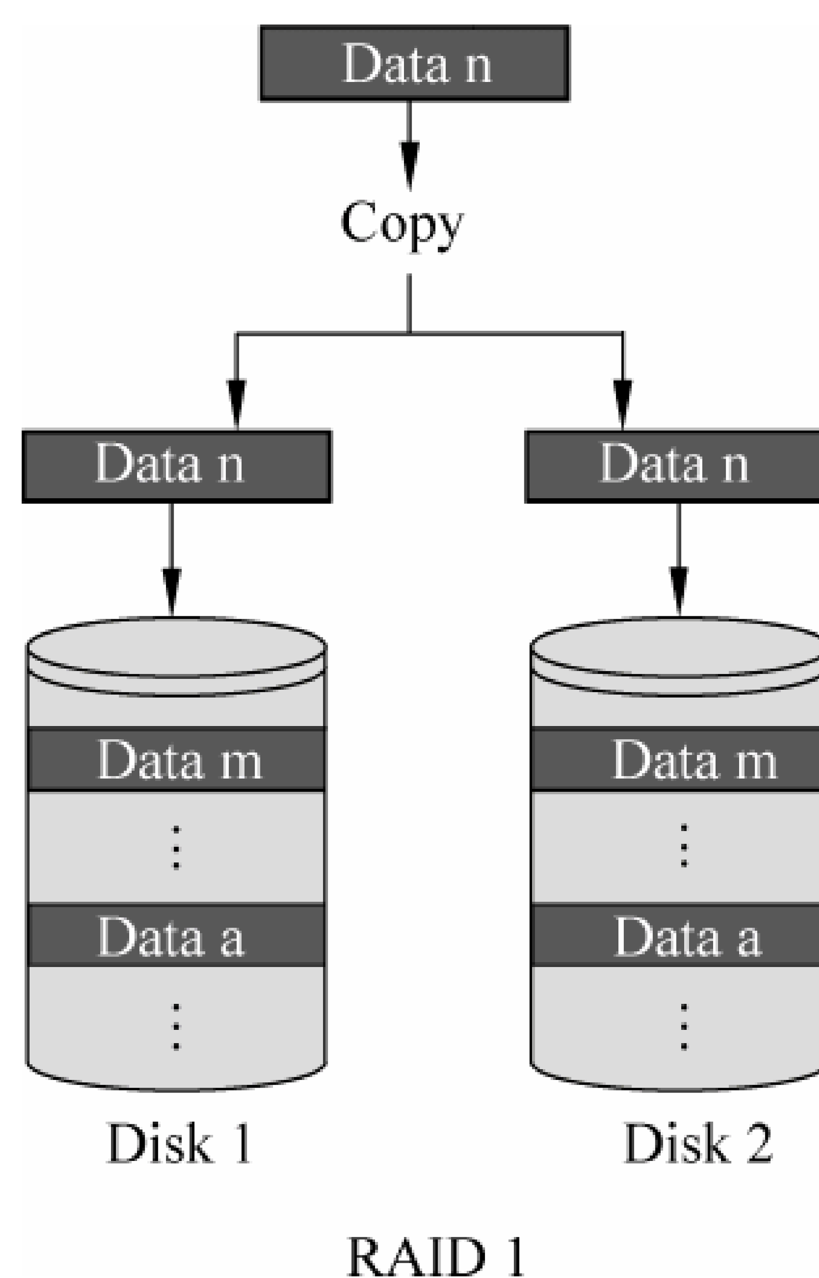


图 7.2　RAID 1 工作原理

- ❑ RAID 1 的可用空间与组成阵列的磁盘中容量最小的磁盘空间相等。
- ❑ RAID 1 不能加快数据的读写速度。

由于每块磁盘内的数据一致，因此通常将 RAID 1 称为镜像卷。

3. RAID 5

组成 RAID 5 必须使用 3 块以上磁盘，其工作原理如图 7.3 所示。

以使用 3 块磁盘组成的 RAID 5 阵列为例：存储数据时，先将数据分为两部分，然后对这两部分数据执行奇偶校验运算（即异或运算），最后将这两部分数据与奇偶校验的结果随机地存放在 3 块磁盘中。

RAID 5 的特性如下。

- ❑ RAID 5 提供磁盘冗余功能。
- ❑ 同时读写磁盘，加快了数据读写速度。
- ❑ RAID 5 需要执行奇偶校验运算，因此要消耗计算资源。
- ❑ RAID 5 的可用空间计算公式为：$n*(m-1)$。其中 n 表示单个磁盘的空间，m 表示组成阵列的磁盘总数。

注意：RAID 5 的冗余工作机制为，当其中一块磁盘损坏时，可以通过其他几块磁盘中的数据计算出丢失的数据。因此虽然 RAID 5 提供了磁盘冗余功能，但如果损坏的磁盘数大于 1，仍然无法计算出丢失的数据。

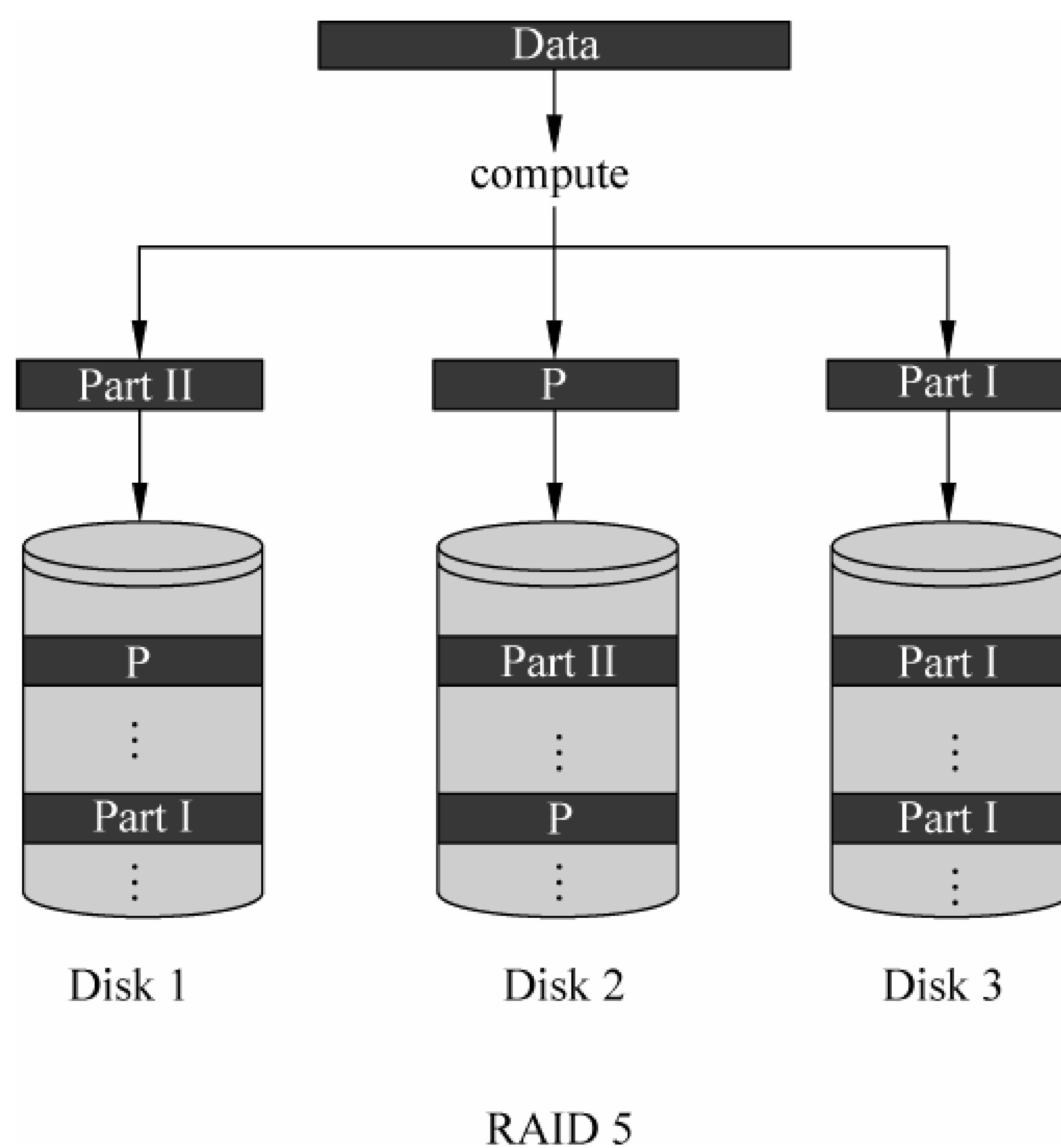


图 7.3　RAID 5 工作原理

本小节介绍的几种磁盘阵列级别中，RAID 5 应用得比较广泛。除此之外，还有一些不常用的磁盘阵列级别，感兴趣的读者可以阅读相关文档，此处不再赘述。

7.3.3　创建组成阵列的磁盘分区

在 Linux 系统中，阵列的实现方式是：利用多个现有的块设备文件，通过操作系统的底层代码虚拟出新的块设备文件（称为虚拟块设备文件）。实现虚拟块设备文件功能的底层代码叫做 MD（Multiple Device），利用 MD 可以组成 RAID 0、RAID 4、RAID 5、RAID 6 等多种阵列。

与 Windows 环境中的软件阵列不同，MD 使用磁盘分区创建阵列（Windows 系统中只能使用磁盘组成阵列）。本小节将介绍如何创建组成阵列的磁盘分区。

下面以创建一个磁盘阵列分区 sdb1 为例，介绍创建分区的过程：

```
#使用 fdisk 命令为磁盘 sdb 分区
# fdisk /dev/sdb
#以下省略的是一些提示信息
The number of cylinders for this disk is set to 10443.
……
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

#使用命令 n 和 p 添加主要分区
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
#新分区使用分区号 1，并使用所有可用空间
Partition number (1-4): 1
First cylinder (1-10443, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-10443, default 10443):
Using default value 10443

#使用命令 t 修改分区类型
Command (m for help): t
Selected partition 1
#分区类型代码为 fd
Hex code (type L to list codes): fd
Changed system type of partition 1 to fd (Linux raid autodetect)

#使用命令 p 查看添加的分区
Command (m for help): p

……
#新建立的分区类型为 Linux raid autodetect
   Device Boot      Start       End      Blocks    Id  System
/dev/sdb1               1      10443    83883366   fd  Linux raid autodetect

#使用命令 w 将分区表同步到磁盘
Command (m for help): w
```

```
The partition table has been altered!
……
Syncing disks.
```

上面这个过程与创建普通磁盘分区类似，但在上面创建分区的过程中，使用了命令 t 将分区类型修改为 Linux raid autodetect。虽然不修改分区类型也可以创建磁盘阵列，但为了方便管理员了解其功能，建议修改组成阵列的所有磁盘分区。

按照上面的步骤依次添加磁盘分区 sdc1 和 sdd1。

提示：使用命令 t 修改分区类型时，如果不知道对应分区的代码，可以使用 L 命令列出所有的分区类型及其对应的代码。

7.3.4　创建磁盘阵列

在 Linux 系统中，创建和管理磁盘阵列使用的命令是 mdadm。由于这个命令的使用方法非常复杂，并且拥有许多选项，因此本书将在示例中介绍这些选项的用法。

在开始创建磁盘阵列之前，需要注意以下两点。

- 虽然系统不会提示，但组成阵列的所有分区大小应该基本一致。最好使用几个相同的磁盘，每个磁盘创建一个分区，然后创建阵列。
- 不要使用一个磁盘的多个分区组建阵列，这会大大影响阵列的读写速度。

除了要注意以上两点之外，建议使用相同型号的磁盘组成阵列（相同型号的磁盘参数都相同），以保证阵列获得最佳的性能。

1．创建磁盘阵列

在本例中，将使用上一小节中创建的 3 个磁盘分区：sdb1、sdc1 和 sdd1，要组成的阵列级别为 RAID 5。

（1）创建阵列的命令如下：

```
#使用 mdadm 创建阵列
#选项 create 表示要创建的磁盘阵列的设备名，此处要创建的是/dev/md0
#选项 level 表示磁盘阵列的级别，本例中为 5
#选项 raid-devices 表示阵列将使用的磁盘分区数，最后是分区列表
#本例中分区数为 3，组成磁盘阵列的分区分别是 sdb1、sdc1、sdd1
# mdadm --create /dev/md0 \
> --level 5 \
> --raid-devices 3 /dev/sdb1 /dev/sdc1 /dev/sdd1
#以下为命令输出的提示信息
mdadm: /dev/sdb1 appears to contain an ext2fs file system
    size=83883364K  mtime=Thu Jan  1 08:00:00 1970
#询问用户是否要继续创建，输入 y 后按 Enter 键即可
Continue creating array? y
mdadm: array /dev/md0 started.
#使用 ls 命令查看新建立的阵列 md0 的块设备文件
# ls -l /dev/md0
brw-r----- 1 root disk 9, 0 Sep 21 19:43 /dev/md0
```

上面这个示例命令中，使用了长格式选项 create、level、raid-devices，其对应短格式选

项为 C、l 和 n。

（2）磁盘阵列创建完成后，系统会立即同步磁盘，并初始化创建的阵列。由于这个过程会读写磁盘，因此通常不建议在此时对阵列执行读写操作。

如果要查看初始化进度，可以使用以下命令：

```
#使用 cat 命令查看/proc/mdstat
# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
#以下为刚刚新建的阵列 md0 的基本情况
md0 : active raid5 sdd1[3] sdc1[1] sdb1[0]
     167766528 blocks level 5, 64k chunk, algorithm 2 [3/2] [UU_]
     [====>................]  recovery = 20.8% (17488924/83883264)
     finish=71.8min speed=15397K/sec

unused devices: <none>
```

命令前两行显示了阵列 md0 的级别、组成阵列的磁盘、阵列的大小等组成情况。后 4 行显示了当前阵列的初始化进度、预期时间等。

2. 写入配置文件

为方便系统跟踪和监视阵列，应该将阵列信息写入配置文件/etc/mdadm.conf 中（默认情况下并不存在这个文件）。

可以使用如下命令写入配置文件：

```
#使用 echo 命令将阵列的组成信息写入配置文件
#如果当前写入的阵列不是系统中的第 1 个阵列，下面的命令应该使用追加方式写入文件
# echo "DEVICE /dev/sdb1 /dev/sdc1 /dev/sdd1">/etc/mdadm.conf
#使用选项 detail 和 scan 命令扫描磁盘阵列，并将详细信息写入配置文件
# mdadm --detail --scan>>/etc/mdadm.conf
#使用 cat 命令查看配置文件内容
# cat /etc/mdadm.conf
DEVICE /dev/sdb1 /dev/sdc1 /dev/sdd1
ARRAY /dev/md0 level=raid5 num-devices=3 spares=1 UUID=8766a907:86b17752:
9ac4922c:97cee755
```

从以上命令输出中可以看到，写入配置文件的信息有：组成阵列的设备、阵列设备名称、阵列级别等。

3. 使用阵列

完成以上步骤后，就可以像使用分区那样使用阵列了：

```
#使用 mkfs.ext3 为阵列 md0 创建文件系统
# mkfs.ext3 /dev/md0
#使用 mount 命令挂载阵列文件系统
# mount /dev/md0 /mnt/md0
```

至此，就可以使用阵列存储数据了。

7.3.5　为阵列添加热备盘

如果使用了具有冗余功能的磁盘阵列，应该考虑为阵列添加热备盘（即阵列的后备磁盘）。当阵列中的某个磁盘损坏时，阵列将自动使用热备盘替代已损坏的磁盘并继续工作。本小节将简单介绍如何添加热备盘。

添加热备盘之前，需要查看磁盘阵列的状态：

```
#通过查看/proc/mdstat 的方法查看阵列状态
# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sdd1[2] sdc1[1] sdb1[0]
      167766528 blocks level 5, 64k chunk, algorithm 2 [3/3] [UUU]

unused devices: <none>
```

在上面的示例命令中，第 2 行中的 active 表示阵列/dev/md0 工作正常。第 3 行中的[3/3]和[UUU]表示组成阵列的磁盘有 3 个，并且 3 个磁盘都处于正常工作状态（“U”表示正常，“_”表示不正常）。

要为阵列添加热备盘，可以使用选项 add（或短格式选项 a）：

```
#使用选项 add 为阵列添加热备盘 sde1
# mdadm /dev/md0 --add /dev/sde1
mdadm: added /dev/sde1
#添加成功后查看阵列状态
# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sde1[3](S) sdd1[2] sdc1[1] sdb1[0]
      167766528 blocks level 5, 64k chunk, algorithm 2 [3/3] [UUU]

unused devices: <none>
```

添加热备盘后，查看阵列信息发现组成阵列的磁盘多了一块，其状态标识为 S，表示这是一个热备盘。

为阵列添加热备盘后，阵列不会立即在热备盘中存放数据。只有当阵列中的磁盘损坏时，阵列才会启用热备盘。

提示：为阵列添加的热备盘的创建方法，与组成阵列的其他磁盘的创建方法一样，并且都要求大小基本一致。

7.3.6　使用热备盘替换损坏磁盘

如果阵列中一个磁盘已经损坏，阵列会自动使用热备盘将已损坏的磁盘替换掉，并重新同步阵列中保存的数据。本小节将模拟实现这个过程。

（1）现在假定构成阵列的设备 sdd1 损坏。此处使用选项 fail（短格式选项为 f），人为标识损坏的磁盘：

```
#使用选项 f 标识已损坏的磁盘设备 sdd1
# mdadm -f /dev/md0 /dev/sdd1
mdadm: set /dev/sdd1 faulty in /dev/md0
#查看阵列的状态
# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sde1[3] sdd1[4](F) sdc1[1] sdb1[0]
#由于新加入的第 3 块磁盘中没有数据，因此以下显示第 3 块磁盘工作不正常
     167766528 blocks level 5, 64k chunk, algorithm 2 [3/2] [UU ]
     [>....................] recovery = 0.1% (143476/83883264) finish=
     106.9min speed=13043K/sec

unused devices: <none>
```

从上面的命令输出中可以看到，设备 sdd1 的状态被设置为 F，并且热备盘已经被自动添加到阵列中，系统正利用保存的奇偶校验信息自动恢复数据。

（2）完成上述操作后，可以将已经损坏的磁盘 sdd1 从阵列中移除。从阵列中移除损坏的磁盘可以使用 remove 选项：

```
#使用选项 remove 从阵列中移除磁盘
# mdadm /dev/md0 --remove /dev/sdd1
mdadm: hot removed /dev/sdd1
```

执行上述命令后，sdd1 就已经从阵列中移除了。

7.3.7　扩展阵列

许多时候已经建立的磁盘阵列不能满足应用需求，这时就需要扩展阵列（即扩充阵列的容量）。扩展阵列的操作步骤是，先将扩展磁盘添加为阵列的热备盘，然后再扩展阵列。

在上例中，已经将热备盘 sdd1 和 sdf1 添加到阵列中，现在需要扩展阵列。扩展现有阵列需要使用选项 grow 和 raid-devices，过程如下：

```
#查看阵列的基本情况
# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
#阵列 md0 中存在两个热备盘 sdf1 和 sdd1
md0 : active raid5 sdf1[3](S) sdd1[4](S) sde1[2] sdc1[1] sdb1[0]
     167766528 blocks level 5, 64k chunk, algorithm 2 [3/3] [UUU]

unused devices: <none>
#使用选项 grow 和 raid-devices 扩展阵列
# mdadm --grow /dev/md0 --raid-devices 5
mdadm: Need to backup 256K of critical section..
mdadm: ... critical section passed.
#查看扩展后的阵列信息
# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
#组成阵列的磁盘由原来的 3 个变成 5 个，并且阵列开始自动同步数据
md0 : active raid5 sdf1[3] sdd1[4] sde1[2] sdc1[1] sdb1[0]
     167766528 blocks super 0.91 level 5, 64k chunk, algorithm 2 [5/5] [UUUUU]
```

```
    [>....................]  reshape =  0.1% (124480/83883264) finish=
    213.0min speed=6551K/sec

unused devices: <none>
```

命令将两个热备盘添加到现有阵列中，构成阵列的磁盘数量从 3 个扩展到 5 个。扩展完成后，系统会自动同步阵列中的数据。

本节使用了带有冗余功能的 RAID 5 作为示例讲解阵列的管理。由于其他阵列级别可能不具备冗余功能，因此本书中列举的某些操作可能并不适用于所有阵列。

提示： 除本书中介绍的磁盘阵列级别之外，还可以将磁盘先组成 RAID 1，然后再将多个 RAID 1 阵列组成 RAID 0，这样既能提高读写速度又具有冗余功能。

7.4　LVM 逻辑卷管理

在前面几节中讲到，要使用磁盘存储数据，必须要先分区、建立文件系统，之后才可以挂载并使用文件系统存储数据。但是一些生产系统凸显了这种方式的弊端：随着应用的不断扩展，存储的数据越来越多（例如数据库、文件服务器等），分区大小不能在线扩充就成了一个棘手的问题。为了解决这个问题，管理员们不得不关闭正在提供的业务，然后在线下执行扩展和数据转移工作。由于数据量巨大，线下扩展通常花费的时间很长，这对于关键性业务而言可能是致命的。

可以使用 LVM 来解决这一难题。LVM 是 Logical Volume Manager 的缩写，中文含义是逻辑卷管理。LVM 可以将多个不同容量的分区合并在一起，然后重新分配使用空间。不仅如此，LVM 还支持在线添加磁盘、删除磁盘、在线扩充文件系统等功能。本节将简单介绍 LVM 的基本概念和实现方法。

7.4.1　LVM 的基本概念

对于许多初学者而言，LVM 有许多容易混淆的概念，学习起来非常麻烦，但只要理解其工作原理，学习难度将会大大降低。本小节将简单介绍 LVM 的基本术语和工作原理。

1．基本术语

LVM 中有许多概念和术语，最基本的概念和术语及其含义如下。

- 物理卷（PV）：PV 是 Physical Volume 的缩写。物理卷是构成 LVM 的基本存储设备。在实际应用中，物理卷可以是磁盘分区、RAID 设备等。
- 卷组（VG）：VG 是 Volume Group 的缩写。卷组就像一个可以扩充的磁盘（可以简单地理解为卷组是未分区的磁盘），由一个或多个物理卷组成。
- 逻辑卷（LV）：LV 是 Logical Volume 的缩写。逻辑卷就像是卷组中的一个分区，可以创建文件系统，挂载并存储数据。

将多个物理卷组成卷组后，就可以在卷组的基础上创建逻辑卷、文件系统并存储数据。由于逻辑卷在卷组的基础上创建，而卷组又由多个物理卷组成，因此一个逻辑卷可能位于

不同的物理卷之上。

在磁盘中，系统会通过查找扇区的方式寻址并存取数据（参考磁盘相关知识）。由于逻辑卷可能会位于不同的磁盘之上，因此不能使用扇区寻址。为解决逻辑卷寻址的问题，LVM 引入了两个新概念：物理块和逻辑块。

- 物理块（PE）：PE 是 Physical Extent 的缩写。加入卷组的物理卷被划分为许多大小相等的物理块，物理块的大小通常为 4MB。物理块的功能类似于磁盘上的扇区，每个物理块都具有唯一的地址，LVM 通过物理块在磁盘中寻址。与磁盘中的扇区类似，物理块也是 LVM 中最小的可寻址存储单元。
- 逻辑块（LE）：LE 是 Logical Extent 的缩写。与物理卷一样，逻辑卷也被划分为许多大小相等的逻辑块，其大小和物理块大小相同。逻辑块是逻辑卷上可寻址的最小存储单元。

逻辑块和物理块之间的关系是一一对应的，即一个逻辑块对应一个物理块。在物理卷的起始处，有一个类似于分区表的区域，称为 VGDA（卷组描述符区域）。LE 和 PE 之间的对应关系就保存在 VGDA 的一个表中，除此之外 VGDA 还保存有卷组描述符等内容。

提示： 读者可以简单地理解为：物理块是物理卷中的“扇区”，逻辑块是逻辑卷中的“扇区”。

2. LVM工作原理

存储数据时，LVM 会为数据分配逻辑块，将数据存储到分配的逻辑块上。此时卷组会分配与逻辑块相对应的物理块，并将数据交给磁盘，磁盘会将数据存储到对应的物理块中。

在这个数据存储的例子中，卷组就像是逻辑卷与物理卷的“中间人”，它负责地址转换工作。读取、写入数据时，卷组会将逻辑卷分配的逻辑块地址转换为物理块地址。

从前面的讲述可以得出 LVM 的分层式逻辑结构，如图 7.4 所示。

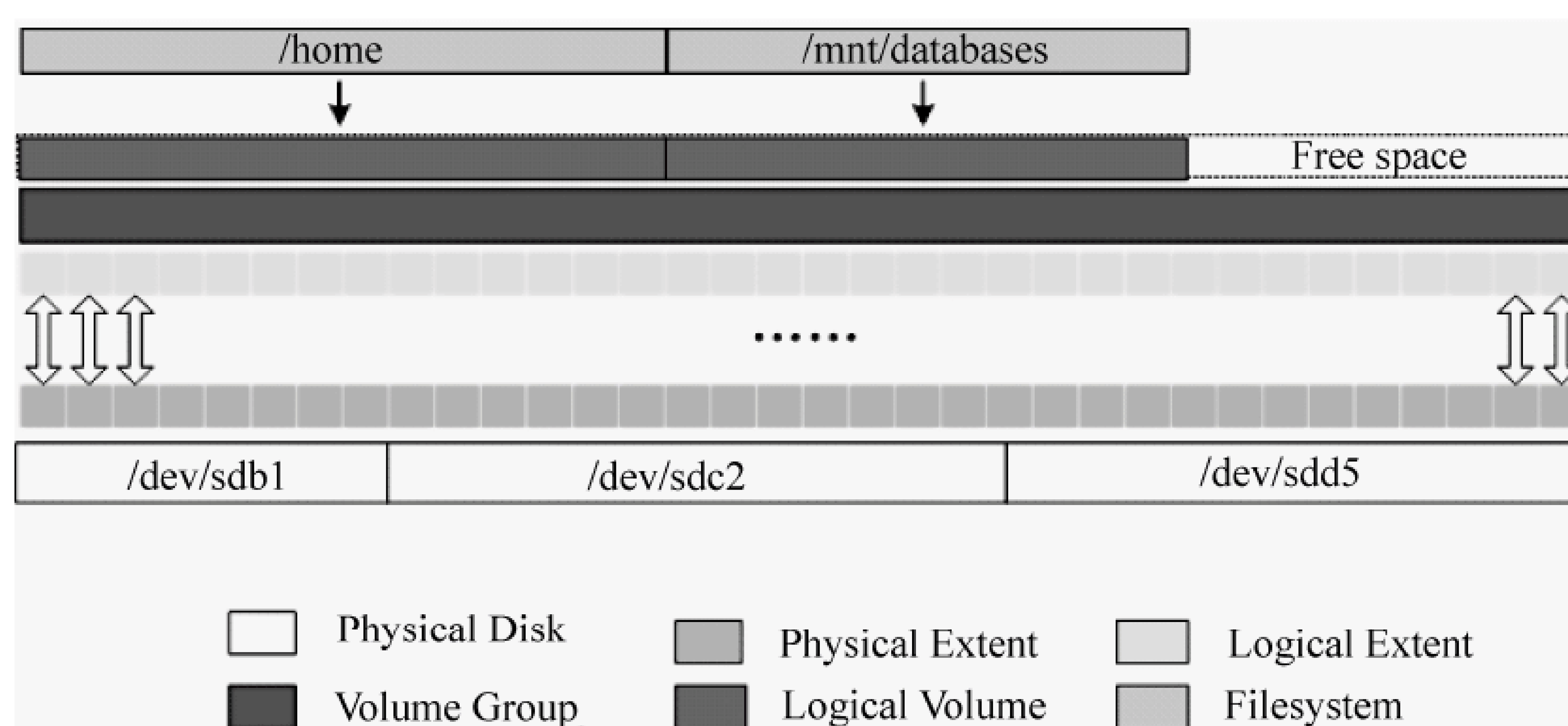


图 7.4　LVM 分层式逻辑结构图

从上面这个结构图中可以清晰地看出，当向目录/home 中写入数据时，系统会首先将数据交给逻辑卷。逻辑卷会为数据分配存储单元（即逻辑块）并存储数据。当卷组收到逻辑卷存储的数据时，会先查询逻辑块与物理块的对应关系，最后再将数据存储到与物理块对应的物理卷中。最终向目录/home 写入的数据，可能会写入 sdb1 中，也可能会写入 sdc2 中，这取决于逻辑卷分配的逻辑块与物理块的对应关系。

了解了以上知识以后，从下一小节开始，将通过一个示例演示如何创建并管理 LVM。

说明：为方便读者理解，本小节中的 LVM 分层式逻辑结构图 7.4 中，组成逻辑卷的物理卷是顺序关系，例如/home 占用了邻近的 sdb1 和 sdc2。这是一个理想状态，实际情况可能并非如此（也有可能会占用 sdb1 和 sdd5），这取决于逻辑卷分配时，逻辑卷的大小和各物理卷的占用情况。

7.4.2　创建物理卷

上一小节介绍了 LVM 的结构，从下到上依次是物理卷、卷组和逻辑卷。创建 LVM 时，也应该按这个顺序依次创建。

物理卷的实质是一个存储设备，通常可以使用的存储设备有磁盘分区、RAID 设备等。创建物理卷的过程，就是向物理卷写入 LVM 信息的过程。本小节将简单讲解如何创建物理卷 PV。

1．创建并修改分区类型

创建物理卷 PV 之前，需要先对磁盘进行分区并修改分区类型，这个过程与创建组成 RAID 设备的分区类似：

```
#使用 fdisk 命令对磁盘 sdb 进行分区
# fdisk /dev/sdb
#由于分区过程与前几节中介绍的过程类似，此处略过
……
#使用命令 t 修改分区号为 1 的分区类型
Command (m for help): t
Selected partition 1
#此处使用的分区类型为 8e
Hex code (type L to list codes): 8e

#使用命令 p 查看上述修改的结果
Command (m for help): p

……
#从分区列表中可以看出分区的类型为 Linux LVM
   Device Boot      Start        End      Blocks       Id  System
/dev/sdb1               1      10443    83883366       8e  Linux LVM

#使用命令 w 同步到磁盘
Command (m for help): w
The partition table has been altered!
……
```

创建分区的过程与前几节中介绍的过程一样，不同的是需要将分区的类型设置为 Linux LVM，以便管理员查看和管理。

本例中还需要依次创建分区 sdc2 和 sdd5。

2．初始化物理卷

分区创建好之后，就可以使用 pvcreate 命令初始化物理卷。pvcreate 命令没有任何选项，直接将要初始化的分区作为其参数即可。

本例中需要初始化的物理卷分别是/dev/sdb1、/dev/sdc2 和/dev/sdd5：

```
#使用 pvcreate 命令初始化 sdb1
# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
#初始化命令 pvcreate 也可以同时初始化多个物理卷
# pvcreate /dev/sdc2 /dev/sdd5
  Physical volume "/dev/sdc2" successfully created
  Physical volume "/dev/sdd5" successfully created
```

虽然本例中使用了 3 个物理卷，但组成卷组的物理卷数量并没有限制，因此实际操作时按需要选择即可。

7.4.3　创建卷组

所有的物理卷都初始化完成后，就可以创建卷组了。创建卷组的命令是 vgcreate，本小节将简单介绍如何创建并查看卷组。

【命令格式】

```
vgcreate vg_name pv
```

在上面的命令格式中，vg_name 是要创建的卷组名称，pv 是组成卷组的物理卷列表。如果列表中有多个物理卷，需要使用空格作为分隔符。

【用法示例】

（1）使用上一小节中初始化的几个物理卷，创建一个名为 LVM01_Volume 的卷组：

```
#使用 vgcreate 命令创建一个名为 LVM01_Volume 的卷组
# vgcreate LVM01_Volume /dev/sdb1 /dev/sdc2 /dev/sdd5
  Volume group "LVM01_Volume" successfully created
```

上面的示例命令中，使用物理卷/dev/sdb1、/dev/sdc2 和/dev/sdd5，创建了一个名为 LVM01_Volume 的卷组。

（2）创建完成后，可以使用命令 vgdisplay 查看新建立的卷组：

```
#使用 vgdisplay 命令查看卷组 LVM01_Volume
#选项 v 表示显示详细信息
# vgdisplay -v LVM01_Volume
    Using volume group(s) on command line
    Finding volume group "LVM01_Volume"
#刚建立的 LVM01_Volume 的基本情况
 --- Volume group ---
  VG Name               LVM01_Volume
  System ID
  Format                lvm2
 ……
```

```
  Act PV                   3
#以下为新创建卷组的容量、PE 大小、剩余空间及分配情况
  VG Size                  202.73 GB
  PE Size                  4.00 MB
  Total PE                 51898
  Alloc PE / Size          0 / 0
  Free  PE / Size          51898 / 202.73 GB
  VG UUID                  M0TLI7-RCnO-pOFr-mM1q-Lo6D-VzpO-8cVtST
#以下为组成卷组的物理卷的基本情况
#包括物理卷名称、状态及空间分配情况等
  --- Physical volumes ---
  PV Name                  /dev/sdb1
  PV UUID                  HLnNvC-9lo3-5dAd-Q45l-3xw9-A30j-mKE5YI
  PV Status                allocatable
  Total PE / Free PE       20479 / 20479
……
```

查看卷组命令 vgdisplay 显示了卷组的详细信息，包括 VG 的名称、大小、PE 的大小和数量等。

提示： 如果需要查看更为详细的卷组信息，可以使用 vgdisplay –vv 或 vgdisplay –vvv 命令。

7.4.4　创建逻辑卷

创建卷组之后，就可以使用 lvcreate 命令在其基础上创建逻辑卷了。在创建逻辑卷之前，还需要规划逻辑卷的大小（按实际需要划分）、逻辑卷的模式等。本小节将简单介绍如何创建逻辑卷。

1．逻辑卷模式

按写入的方式不同，逻辑卷有如下两种模式。

- 线性模式：先写满组成线性逻辑卷的第 1 个物理卷，再向第 2 个物理卷中写入数据，依此类推。
- 交错模式：写入数据时交错地将数据写入组成逻辑卷的多个物理卷中（写入方式类似于 RAID 0）。

从对两种模式的描述中不难看出，交错模式的理论写入速度比线性模式快。由于线性模式与交错模式写入数据的方式不同，因此在使用时，应该按需要选择合适的模式。

2．创建逻辑卷

创建逻辑卷之前，需要先规划逻辑卷的大小及逻辑卷模式。选择逻辑卷模式时，如果逻辑卷需要处理大量读写操作，通常建议使用交错模式，否则应该使用线性模式。

（1）创建一个名为 file，大小为 100GB 的线性逻辑卷：

```
#使用 lvcreate 命令在卷组 LVM01_Volume 上创建一个名为 file 的逻辑卷
#选项 L 用于指定逻辑卷的大小
#选项 n 用于指定逻辑卷的名称
```

```
# lvcreate -L 100G -n file LVM01_Volume
  Logical volume "file" created
```

从上面的命令输出中可以看出，名为 file 的逻辑卷已经创建成功。

（2）如果要创建交错逻辑卷，可以使用以下命令：

```
#使用 lvcreate 命令在卷组 LVM01_Volume 上创建一个名为 test 的交错逻辑卷
#选项 i 表示交错值为 2
#选项 I（大写的字母 i）表示指定逻辑卷的块大小，本例中为 4MB
#选项 l（小写的字母 L）表示块数量，本例中表示块数量为 100 个
# lvcreate -i 2 -I 4 -l 100 -n test LVM01_Volume
  Logical volume "test" created
```

在上面的示例命令中，由于创建时指定为 100 个块，因此逻辑卷 test 的容量只有 400MB。

3．查看逻辑卷

（1）新建的逻辑卷已经放在目录/dev 中，可以使用 ls 命令查看逻辑卷的块设备文件：

```
#使用 ls 命令查看新创建的逻辑卷的块设备文件
#从命令输出中可以看出查看的只是一个链接文件
# ls -l /dev/LVM01_Volume/file
lrwxrwxrwx 1 root root 29 Oct  3 23:46 /dev/LVM01 Volume/file ->
 /dev/mapper/LVM01 Volume-file
#查看真正的逻辑卷块设备文件
# ls -l /dev/mapper/LVM01_Volume-file
brw-rw---- 1 root disk 253, 0 Oct  3 23:46 /dev/mapper/LVM01_Volume-file
```

从上面的命令输出中可以看出，逻辑卷文件以/dev/VGname/LVname 的格式命名。文件/dev/LVM01_Volume/file，其实就是块设备文件/dev/mapper/LVM01_Volume-file 的一个链接文件。

（2）如果要查看指定的逻辑卷详细信息，可以使用命令 lvdisplay：

```
#使用命令 lvdisplay 查看新建立的逻辑卷
#选项 v 表示显示详细信息
# lvdisplay -v /dev/LVM01_Volume/file
#详细信息中包含所属卷组、状态、逻辑卷大小等内容
    Using logical volume(s) on command line
  --- Logical volume ---
  LV Name                /dev/LVM01_Volume/file
  VG Name                LVM01_Volume
  LV UUID                12arhO-UZsO-Vg5k-ABa4-Jc6L-ddFK-yq0sVk
  LV Write Access        read/write
  LV Status              available
  # open                 0
  LV Size                100.00 GB
  Current LE             25600
  Segments               2
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0
```

从上面的命令输出中可以看出逻辑卷的大小、状态等信息。

提示：与查看卷组命令一样，查看更详细的逻辑卷信息，也可以使用 lvdisplay –vv，甚至 lvdisplay –vvv 命令。

4．使用逻辑卷

逻辑卷创建完成之后，其使用方法与分区相同，先创建文件系统，然后挂载并存储数据。

（1）为逻辑卷 file 创建 ext3 文件系统，并挂载到目录/mnt/file：

```
#使用 mkfs.ext3 为新创建的逻辑卷创建文件系统
# mkfs.ext3 /dev/LVM01_Volume/file
#创建挂载点并将逻辑卷挂载到目录/mnt/file
# mkdir /mnt/file
# mount /dev/LVM01_Volume/file /mnt/file/
```

（2）为了能在系统启动时自动挂载逻辑卷，可以在配置文件/etc/fstab 的最后加入以下内容：

```
/dev/LVM01_Volume/file  /mnt/file             ext3   defaults      0 2
```

完成上述步骤之后，就可以使用逻辑卷存储数据了。

注意：虽然新建的逻辑卷在目录/dev 中是一个链接文件，但帮助文档建议使用格式为/dev/VGname/LVname 的链接文件，否则可能会出现意外的错误。

7.4.5　添加物理卷

随着逻辑卷中存储的数据越来越多，其可用空间可能会不足，由于 LVM 支持在线扩展，可以使用这一功能在线扩充空间。但在扩展逻辑卷之前，需要查看卷组中是否还有未分配的空间（查看未分配的空间可以使用 7.4.3 节中介绍的 vgdisplay 命令）。如果空间不足，就需要为卷组添加物理卷。本小节将简单介绍如何添加物理卷。

添加物理卷时，应该先对物理卷进行初始化，之后再利用 vgextend 命令为卷组添加物理卷。

（1）例如为 LVM01_Volume 卷组添加新物理卷/dev/sde3：

```
#先使用 pvcreate 初始化物理卷
# pvcreate /dev/sde3
  Physical volume "/dev/sde3" successfully created
#使用 vgextend 命令将物理卷/dev/sde3 添加到卷组 LVM01 Volume
# vgextend LVM01_Volume /dev/sde3
  Volume group "LVM01_Volume" successfully extended
```

从上述命令的执行结果可以看出，新的物理卷已经成功添加到卷组中。

（2）除命令输出之外，还可以使用命令 vgdispaly，查看物理卷是否添加成功：

```
#使用 vgdisplay 命令查看卷组信息
# vgdisplay
```

```
  --- Volume group ---
  VG Name               LVM01_Volume
  System ID
  Format                lvm2
……
#注意以下内容的变化情况
  VG Size               282.72 GB
  PE Size               4.00 MB
  Total PE              72377
  Alloc PE / Size       25600 / 100.00 GB
  Free  PE / Size       46777 / 182.72 GB
  VG UUID               M0TLI7-RCnO-pOFr-mM1q-Lo6D-VzpO-8cVtST
```

从卷组的大小、及物理块的总量可以看出，物理卷已经添加成功。

7.4.6　扩充逻辑卷

如果卷组还有未分配的空间，就可以使用 lvextend 命令扩充逻辑卷了。扩充逻辑卷时，不必卸载已挂载的文件系统，也无须停止对逻辑卷的读写操作。

（1）要将当前的逻辑卷/dev/LVM01_Volume/file 扩展到 120GB 可以使用如下命令：

```
#使用命令 lvextend 的选项 L 为逻辑卷扩容
# lvextend -L 120G /dev/LVM01_Volume/file
  Extending logical volume file to 120.00 GB
  Logical volume file successfully resized
```

注意上面的示例命令的作用是将逻辑卷 file 扩展到 120GB，而不是为其增加 120GB。

（2）如果需要指定增加空间的大小，可以使用以下命令：

```
#使用选项 L 时使用加号表示在原有基础上增加空间
# lvextend -L +10G /dev/LVM01_Volume/file
  Extending logical volume file to 130.00 GB
  Logical volume file successfully resized
```

此时逻辑卷 file 的容量增加了 10GB，扩展到了 130GB。

（3）细心的用户可能会发现，虽然逻辑卷已经扩充到 130GB，但使用 df –h 命令（该命令用于查看已挂载的文件系统的使用情况）显示的结果却没有变化。原因在于，虽然逻辑卷的容量已经改变，但逻辑卷上的文件系统却没有发生变化，因此还需要在线扩充文件系统。在线扩充文件系统的命令为 resize2fs。

注意：命令 resize2fs 只能用于调整 ext2、ext3 文件系统，如果逻辑卷使用的是其他文件系统，就需要参阅相关文档了解具体的命令。

例如将逻辑卷 file 的文件系统在线扩充至 130GB：

```
#使用 resize2fs 命令对逻辑卷/dev/LVM01_Volume/file 执行在线扩充
# resize2fs /dev/LVM01_Volume/file 130G
resize2fs 1.39 (29-May-2006)
Filesystem at /dev/LVM01 Volume/file is mounted on /mnt/file; on-line
resizing required
```

```
Performing an on-line resize of /dev/LVM01_Volume/file to 34078720 (4k)
blocks.
The filesystem on /dev/LVM01_Volume/file is now 34078720 blocks long.
```

从上面的命令输出中可以看出，resize2fs 命令已经成功地执行了在线扩充，接下来执行 df –h 命令就可以看到正确的容量了。

7.4.7　减小逻辑卷

有时不合理地分配了逻辑卷的容量，导致其他逻辑卷空间不足，也可能是调整空间以便于移除某个物理卷。这时就需要减小某个逻辑卷的空间。减小逻辑卷的空间可以使用命令 lvreduce。本小节将简单介绍如何减小逻辑卷。

相对于逻辑卷扩充过程而言，减小逻辑卷容量的过程比较麻烦。这是因为减小容量的过程中，需要移动某些数据，并修改文件信息。

（1）由于要修改文件信息，因此整个操作过程不能在线进行。首先需要卸载文件系统，并强制检查文件系统是否存在错误：

```
#使用 umount 命令卸载已经挂载的文件系统
# umount /mnt/file/
#使用 fsck 检查文件系统中的错误并修复
#选项 f 表示强制检查
# fsck -f /dev/LVM01_Volume/file
fsck 1.39 (29-May-2006)
e2fsck 1.39 (29-May-2006)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/LVM01_Volume/file: 11/17039360 files (9.1% non-contiguous), 582743/
34078720 blocks
```

由于 fsck 命令会扫描所有数据块，因此当文件系统较大时，可能需要花费大量的时间。

（2）与扩充逻辑卷的过程相反，减小逻辑卷时，应该首先减小文件系统。减小文件系统仍然使用命令 resize2fs。

例如将逻辑卷 file 的文件系统减小至 50GB：

```
#使用 resize2fs 命令将逻辑卷的文件系统减小至 50GB
# resize2fs /dev/LVM01_Volume/file 50GB
resize2fs 1.39 (29-May-2006)
Resizing the filesystem on /dev/LVM01_Volume/file to 13107200 (4k) blocks.
The filesystem on /dev/LVM01_Volume/file is now 13107200 blocks long.
```

由于 resize2fs 命令会合并未使用空间并移动数据，因此上面这个示例命令可能需要花费许多时间。

提示：减小文件系统之前，应该保证逻辑卷中存放的数据减小后的空间，否则可能会损坏逻辑卷中的数据。

（3）文件系统减小后，就可以使用 lvreduce 命令减小逻辑卷的容量了：

```
#使用 lvreduce 命令将逻辑卷减小至 50GB
#选项 L 用于指定减小后的逻辑卷大小
# lvreduce -L 50G /dev/LVM01_Volume/file
  /dev/cdrom: open failed: Read-only file system
  Attempt to close device '/dev/cdrom' which is not open.
  WARNING: Reducing active logical volume to 50.00 GB
  THIS MAY DESTROY YOUR DATA (filesystem etc.)
#要求用户确认减小逻辑卷容量，本例中输入 y，按 Enter 键
Do you really want to reduce file? [y/n]: y
  Reducing logical volume file to 50.00 GB
  Logical volume file successfully resized
```

命令会询问用户是否需要减小逻辑卷的大小，确认之后系统就会减小逻辑卷。

完成上述操作后，就可以重新挂载文件系统，并使用命令 df –h 验证结果。

7.4.8　移动数据并移除物理卷

有时需要将物理卷从卷组中删除，以便于添加更大容量的磁盘，以扩充逻辑卷的存储空间。此时可以使用 pvremove 命令。本小节将简单介绍如何移动物理卷中的数据，并移除物理卷。

移除物理卷时，一定要确保卷组中未分配的连续可用物理块，大于移除的物理卷中分配的物理块。只有这样，才能将物理卷中的数据移动到其他物理卷中。

（1）在本例中，将演示如何从卷组中移除物理卷/dev/sdd5。首先需要使用 pvdispaly 查看物理卷的分配情况：

```
#使用 pvdisplay 命令查看物理卷的分配情况
#选项 m 用于显示 PE 到 LV、LE 之间的对应关系
# pvdisplay -m
#组成卷组的物理卷的基本情况
  --- Physical volume ---
  PV Name               /dev/sdb1
  VG Name               LVM01 Volume
  PV Size               80.00 GB / not usable 1.35 MB
  Allocatable           yes (but full)
  PE Size (KByte)       4096
#从下面这两行可以看出，sdb1 中的所有 PE 都已经被分配
  Total PE              20479
  Free PE               0
……
#以下是物理卷与逻辑卷的对应关系
#从以下关系中可以看出 sdb1 对应的所有 LE 都被分配给逻辑卷/dev/LVM01 Volume/file
  --- Physical Segments ---
  Physical extent 0 to 20478:
    Logical volume      /dev/LVM01 Volume/file
    Logical extents     0 to 20478

  --- Physical volume ---
```

```
  PV Name               /dev/sdc2
  VG Name               LVM01_Volume
  PV Size               42.74 GB / not usable 3.01 MB
  Allocatable           yes
  PE Size (KByte)       4096
#从以下几行可以看出物理卷 sdc2 没有被分配，所有 PE 可用
  Total PE              10940
  Free PE               10940
……
#以下 3 行表示没有任何逻辑卷使用物理卷 sdc2
  --- Physical Segments ---
  Physical extent 0 to 10939:
    FREE

#由于要移动 sdd5 中的数据，因此要特别注意 sdd5 的分配情况
  --- Physical volume ---
  PV Name               /dev/sdd5
  VG Name               LVM01_Volume
  PV Size               80.00 GB / not usable 1.32 MB
  Allocatable           yes
  PE Size (KByte)       4096
#注意 sdd5 中已经分配的 PE 数目
  Total PE              20479
  Free PE               10238
  Allocated PE          10241
  PV UUID               L3vuZF-W6XV-nkOO-vpfx-MOsI-yRzw-TGKCDv

#注意使用物理卷的逻辑卷为/dev/LVM01_Volume/file
  --- Physical Segments ---
  Physical extent 0 to 10240:
    Logical volume      /dev/LVM01_Volume/file
    Logical extents     20479 to 30719
  Physical extent 10241 to 20478:
    FREE

  --- Physical volume ---
  PV Name               /dev/sde3
  VG Name               LVM01_Volume
  PV Size               80.00 GB / not usable 1.35 MB
  Allocatable           yes
  PE Size (KByte)       4096
#物理卷 sde3 未被分配，所有 PE 可用
  Total PE              20479
  Free PE               20479
  Allocated PE          0
  PV UUID               W1mGDh-EamE-dRPl-YaMW-7AvX-hlVO-s85gLv
……
```

从上面的命令输出中可以看出，物理卷/dev/sdd5 中，已经被占用的物理块共有 10 241 个。因此移动数据时，必须要保证目标物理卷的可用 PE 大于 10 241。满足此条件的物理卷有/dev/sdc2 和/dev/sde3，因此可以将/dev/sdd5 中占用的物理块移动到这两个物理卷中。

（2）确认数据移动的目标物理卷后，就可以使用 pvmove 命令移动 PE 了。例如将/dev/sdd5 上的数据移动到/dev/sde3 中：

```
#使用 pvmove 命令移动物理卷中的数据
# pvmove /dev/sdd5 /dev/sde3
  /dev/sdd5: Moved: 3.4%
  /dev/sdd5: Moved: 9.5%
  ……
  /dev/sdd5: Moved: 100.0%
```

移动数据时，命令会显示当前移动数据的进度。由于物理卷中可能会有很多数据，因此这个过程通常会很慢，建议在逻辑卷读写压力较小时进行。

移动物理卷数据的过程中，一定要确保没有新的逻辑卷被分配，否则可能又会占用要移除的物理卷。

如果卷组中的剩余 PE 足够，也可以使用以下命令移动数据：

```
#使用 pvmove 命令自动移动物理卷/dev/sdd5 中的数据
#选项 i 表示每移动指定百分比的数据，就向用户报告
# pvmove -i 10 /dev/sdd5
```

此命令也会移动物理卷/dev/sdd5 上的数据，不同的是此命令将由系统自己决定将数据移动到哪个物理卷上。

（3）确认没有逻辑卷占用物理卷后，可以使用命令 vgreduce 将其移出卷组。例如移除物理卷/dev/sdd5：

```
#使用 vgreduce 命令将物理卷/dev/sdd5 从卷组 LVM01 Volume 中移除
# vgreduce LVM01 Volume /dev/sdd5
  Removed "/dev/sdd5" from volume group "LVM01_Volume"
```

此时物理卷/dev/sdd5 就被移出卷组了。

注意：移除物理卷操作非常危险，应该确认所有的数据均已移动到其他物理卷上，否则会造成数据丢失。

7.4.9　逻辑卷快照

对数据库文件进行备份时，由于数据库处于工作状态，因此不停地写入、删除数据库中的信息，会造成备份后的数据前后不一致。这时可以使用快照（snapshot），其原理如同这个名词本身的含义一样，是将存储数据的逻辑卷做一个只读的拷贝。利用这个只读的拷贝进行备份，就可以解决数据前后不一致的问题。

创建逻辑卷快照与创建逻辑卷一样，都使用 lvcreate 命令。

（1）创建快照卷之前，需要确认原始卷的大小，因为创建的快照卷通常都应该比原始卷大（1.1 倍至 1.2 倍之间即可）。

例如要为/dev/LVM01_Volume/databases 创建快照卷：

```
#使用 lvcreate 命令创建名为 db_snapshot 的快照卷
#选项 L 用于指定快照卷的大小
#选项 s 表示要创建的是快照卷
```

```
#选项 n 用于指定快照卷的名称
# lvcreate -L 40G -s -n db_snapshot /dev/LVM01_Volume/databases
  Logical volume "db_snapshot" created
```

上面这个命令为逻辑卷/dev/LVM01_Volume/databases 创建了一个名为 db_snapshot，大小为 40GB 的快照卷。

快照卷创建完成后，就可以挂载并备份其中的数据文件了。

（2）完成备份之后，可以使用以下命令卸载并删除快照卷：

```
#使用 umount 命令卸载快照卷
# umount /dev/LVM01_Volume/db_snapshot
#使用 lvremove 命令删除快照卷
# lvremove /dev/LVM01_Volume/db_snapshot
#命令要求用户确认删除快照卷
Do you really want to remove active logical volume "db_snapshot"? [y/n]: y
  Logical volume "db_snapshot" successfully removed
```

执行上面的命令后，快照卷就被删除了。

快照卷具有非常广泛的用途，例如快速还原虚拟机，其实现的原理也非常复杂，本书对此不做介绍，感兴趣的读者可以阅读相关文档。

7.5　磁盘配额管理

在多用户系统中，如果没有对用户使用的磁盘空间做出限制，用户无限制地存放数据和文件，可能会导致系统磁盘空间告警。如果存放的是无用数据，就会导致磁盘空间白白浪费。磁盘配额可以限制用户或组在磁盘上存放文件的空间，这样既可以让用户拥有一定的存储空间，又可以避免用户无限制地存放数据导致系统磁盘空间不足。

磁盘配额在多用户系统、文件服务器、虚拟机中均有广泛的用途。本节将简单介绍如何使用磁盘配额管理用户的磁盘空间。

7.5.1　为磁盘配额提供支持

在 Linux 系统中，为磁盘配额提供支持的是 quota 软件包，如果系统没有安装这个软件包，用户可以自行安装。安装软件包的方法将在第 9 章中介绍。

在设置磁盘配额之前，必须要获得文件系统的支持。为此需要系统中的自动挂载文件/etc/fstab，为相应的文件系统添加挂载参数：usrquota（启用用户限额）和 grpquota（启用用户组限额）。

此处将以磁盘分区/dev/sdd1 为例，讲解如何建立磁盘配额。

（1）要启用文件系统 sdd1 的配额支持，将其修改为以下内容：

```
/dev/sdd1       /mnt/sdd1       ext3    defaults,usrquota,grpquota      0  0
```

（2）修改完成之后，需要重新挂载文件系统，以启动磁盘配额支持：

```
#使用 umount 及 mount 命令重新挂载文件系统
```

```
# umount /mnt/sdd1
# mount /dev/sdd1 /mnt/sdd1
```

也可以使用以下命令重新挂载/etc/fstab 中记录的所有分区：

```
#使用 mount 的选项 a 重新挂载所有分区
# mount -a
```

（3）修改完成后，可以使用 mount 命令验证设置：

```
#使用 mount 命令查看挂载的分区及参数
# mount
……
/dev/sdd1 on /mnt/sdd1 type ext3 (rw,usrquota,grpquota)
```

如果在命令输出中看到 usrquota 和 grpquota，表明已经成功启用磁盘配额支持。

7.5.2　检查磁盘配额命令 quotacheck

启用了文件系统的配额支持后，还不能立即使用配额，还必须检查相应的文件系统，并建立磁盘配额文件。检查文件系统并建立磁盘配额文件，需要使用命令 quotacheck。本小节将简单介绍如何使用 quotacheck 命令建立磁盘配额文件。

【命令格式】

```
quotacheck [option] filesystem
```

【常用选项】

- ❑ c：跳过原有配额文件，只执行新的扫描并保存到磁盘。
- ❑ v：显示命令详细信息。
- ❑ u：检查用户使用文件系统的情况。
- ❑ g：检查用户组使用文件系统的情况。

除了以上这些选项外，quotacheck 还有许多不常用的选项，读者可以参考其帮助文档。

【用法示例】

（1）检查磁盘分区/dev/sdd1，并将相关信息写入到配额文件中：

```
#使用 quotacheck 命令为文件系统建立磁盘配额文件
# quotacheck -cug /mnt/sdd1
```

由于 quotacheck 命令会检查和测试整个文件系统，因此可能需要花费较长时间。

（2）如果要验证上述命令的执行结果，可以使用 ls 命令查看建立的配额配置文件：

```
#使用 ls 命令查看磁盘配额的配置文件
# ls -l /mnt/sdd1/aquota*
-rw------- 1 root root 7168 Oct  4 22:45 /mnt/sdd1/aquota.group
-rw------- 1 root root 7168 Oct  4 22:45 /mnt/sdd1/aquota.user
```

从上面的命令输出中可以看到，命令在文件系统的根目录中建立了两个文件：aquota.group 和 aquota.user。这两个文件分别用于存放用户和用户组的磁盘配额信息，包括文件系统的使用情况及配置信息等。

7.5.3　查看磁盘使用情况命令 repquota

在所有用户都可以使用的文件系统上，并不需要为所有用户都设置磁盘配额。通常只需要为那些经常存放大文件、严重占用磁盘空间的用户设置即可。

查看文件系统的使用情况，可以使用 **repquota** 命令（使用该命令的前提是已经创建磁盘配额文件）：

```
#使用 requota 命令查看文件系统/mn/sdd1 的使用情况
# repquota /mnt/sdd1
*** Report for user quotas on device /dev/sdd1
Block grace time: 7days; Inode grace time: 7days
#以下为上一小节中统计的文件系统使用情况
#使用情况分为两个部分，块限制部分和文件限制部分
                    Block limits              File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --  282412    0         0            5     0     0
wlh       --  342232    0         0            1     0     0
```

查看上述命令的输出并选择要限制的用户，之后就可以建立配额限制了。

7.5.4　建立磁盘配额命令 edquota

完成前几小节的准备工作后，就可以开始设置硬盘配额了。建立磁盘配额需要使用 edquota 命令，本小节将通过示例讲解如何为用户和用户组建立磁盘配额。

【命令格式】

```
edquota [option] [username|groupname]
```

使用 edquota 命令建立磁盘配额时，需要指定创建配额的用户和用户组名。

【常用选项】

- ❑ u：指定限制的用户名称。
- ❑ g：指定限制的用户组名称。
- ❑ p：复制用户或组的配额信息。
- ❑ t：修改过渡期，即用户的磁盘配额超过软限制的宽限时间。

【用法示例】

（1）以为用户 wlh 建立配额为例：

```
#使用选项 u 为用户 wlh 建立磁盘配额
# edquota -u wlh
```

此时命令会启动用户配额配置界面，配置界面文本如下：

```
Disk quotas for user wlh (uid 502):
  Filesystem         blocks      soft     hard         inodes     soft      hard
  /dev/sdd1          342232       0        0            1          0         0
```

此时细心的读者可以发现，其实这个界面就是 Vi 编辑器，这时就可以像使用 Vi 那样编辑这些设置（Vi 编辑器将在第 11 章中介绍，不了解的读者可以先阅读该部分内容）。

配置文本一共有 7 个字段，各字段及其含义如下。

- Filesystem：当前正在设置配额限制的文件系统。
- blocks：该字段表示当前用户已经使用的磁盘分区空间，单位是 KB。
- soft：对用户使用的磁盘空间或文件数的软限制（磁盘空间单位是 KB）。用户使用的磁盘空间或文件数在过渡期内可以超过这个限制。
- hard：表示对用户使用磁盘空间或文件数的硬限制。用户使用的磁盘空间或文件数绝对不能超过此限制。
- inodes：用户在当前磁盘空间中占用的 i 结点数，由系统自动统计，用户不必修改此项。

在磁盘配额界面中有两个 soft 和 hard 字段，其中前一组 soft、hard 表示对空间大小的限制，后一组表示对文件数的限制。

由于文件数限制对一般的使用者而言意义不大，因此设置磁盘配额时，一般不在文件数上设置限制。

注意：修改磁盘配额时，无论是磁盘空间限制，还是文件数限制，硬限制的数值都应该比软限制大。

在此示例中，假定需要设置用户 wlh 的磁盘空间软限制为 1.5GB，硬限制为 2GB。可以将配额修改为如下内容：

```
Disk quotas for user wlh (uid 502):
  Filesystem         blocks     soft        hard     inodes      soft      hard
  /dev/sdd1          342232   1500000     2000000       1           0         0
```

保存退出后即可完成设置。

设置上述配额限制后，如果 wlh 在该文件系统上使用的空间超过 1.5GB，系统就会警告用户。当用户数据达到 2GB 时，将不能存入任何信息。

（2）如果需要查看设置的磁盘配额，可以使用 quota 命令。例如查看用户 wlh 的磁盘配额限制：

```
#使用 quota 命令查看用户的磁盘配额限制
#选项 u 表示查看指定的用户
# quota -u wlh
Disk quotas for user wlh (uid 502):
     Filesystem  blocks   quota   limit  grace  files  quota  limit  grace
      /dev/sdd1  342232  1500000 2000000             1        0      0
```

（3）如果要为多个用户设置相同的磁盘配额，可以先设置其中一个用户的磁盘配额，然后使用选项 p 将配额信息复制给其他用户。

例如将用户 wlh 的磁盘配额复制给用户 user1 和 user：

```
#使用选项 p 复制磁盘配额设置
# edquota -p wlh -u user1 user
```

（4）如果需要对某个用户组设置磁盘配额，可以使用编辑配额命令 edquota 的选项 g。例如要设置用户组 teacher 的磁盘配额：

```
#使用选项 g 指定磁盘配额的用户组
# edquota -g teacher
```

用户组磁盘配额设置与用户的磁盘配额方法相同，此处不再赘述。

可以使用如下命令查看用户组的磁盘配额：

```
#使用 quota 命令的选项 g 查看用户组的磁盘配额
# quota -g tercher
```

（5）过渡期也称为配额宽限时间，是指用户在磁盘上存放数据的空间或文件数超过了软限制，但没有达到硬限制时，用户还能使用的最长宽限时间。用户应该在过渡时间内，将磁盘使用空间降低到软限制以下，否则将法正常使用磁盘。

使用命令 edquota 的选项 t 可以修改过渡期：

```
#使用选项 t 修改过渡期
# edquota -t
```

此命令将会启动修改过渡期界面。与修改磁盘限额一样，这个界面也是一个 Vi 编辑器界面。

修改过渡期界面显示的文本如下：

```
Grace period before enforcing soft limits for users:
Time units may be: days, hours, minutes, or seconds
  Filesystem             Block grace period     Inode grace period
  /dev/sdd1                     7days                  7days
```

上面的例子已将过渡期设置为 7 天，读者可以自行修改磁盘空间限制（Block grace period）和文件数限制（Inode grace period）的过渡期时间。过渡期的单位可以是 days（天）、hours（小时）、minutes（分钟）和 seconds（秒）。

注意：不要对根分区使用配额限制，否则会导致一些意外的情况发生，例如无法登录系统等。也不要对 root 用户做配额限制，以免发生意外。

7.5.5　开启磁盘配额命令 quotaon

设置好用户、用户组配额和过渡期之后，还需要使用 quotaon 命令，在文件系统上手动开启配额限制功能。

【命令格式】

```
quotaon [option] filesystem
```

【常用选项】

- a：开启所有文件系统的磁盘配额功能。
- u：开启用户磁盘配额。
- g：开启用户组磁盘配额。

- ❑ v：如果开启成功，就显示提示信息。

上面这几个选项中，选项 u 和 g 是默认使用的选项。

【用法示例】

（1）开启指定文件系统的磁盘配额功能：

```
#使用 quotaon 命令开启文件系统/mnt/sdd1 的磁盘配额功能
# quotaon -vug /mnt/sdd1
/dev/sdd1 [/mnt/sdd1]: group quotas turned on
/dev/sdd1 [/mnt/sdd1]: user quotas turned on
```

（2）也可以使用选项 a 开启所有文件系统的磁盘配额功能：

```
#使用选项 a 开启所有文件系统的磁盘配额功能
# quotaon -av
/dev/sdc1 [/mnt/sdd1]: group quotas turned on
/dev/sdc1 [/mnt/sdd1]: user quotas turned on
```

需要注意的是，有些系统可能会自动开启配额，不需要管理员手动开启。

7.5.6　关闭磁盘配额命令 quotaoff

有时可能需要临时关闭磁盘配额功能，例如修改了用户的磁盘配额，需要重新开启等。这时可以使用 quotaoff 命令。

关闭磁盘配额命令 quotaoff 的用法、常用选项与 quotaon 一样，此处不再赘述。

（1）例如关闭文件系统/mnt/sdd1 的磁盘配额功能：

```
#使用 quotaoff 命令关闭文件系统/mnt/sdd1 的磁盘配额功能
# quotaoff -vug /mnt/sdd1/
/dev/sdc1 [/mnt/sdd1]: group quotas turned off
/dev/sdc1 [/mnt/sdd1]: user quotas turned off
```

（2）关闭所有文件系统的磁盘配额功能：

```
#使用选项 a 表示关闭所有文件系统的磁盘配额功能
# quotaoff -av
```

7.5.7　管理磁盘配额

磁盘配额建立之后，管理员通常需要每隔一段时间对设置的磁盘配额进行管理。管理内容包括查看磁盘使用情况，校准磁盘使用信息等。本小节将简单介绍磁盘配额的管理。

1．查看磁盘使用情况

查看磁盘的使用情况如下：

```
#使用 repquota 命令查看磁盘使用情况
# repquota /mnt/sdd1
*** Report for user quotas on device /dev/sdd1
Block grace time: 7days; Inode grace time: 7days
```

```
                    Block limits                File limits
User          used     soft          hard       grace     used  soft  hard  grace
----------------------------------------------------------------------
root    --   282412          0          0                  5      0     0
user1   +-   1742628   1500000    2000000    6days          9      0     0
wlh      --  342232    1500000    2000000                  1      0     0
```

上面的命令输出中，用户名后面的两个减号“--”，用于标识用户当前是否已经超出限制。其中用户 user1 使用的空间已经超出了软限制，此时前面的减号“-”变成了加号“+”。其后的几个字段显示用户的使用情况、限制情况等。grace 字段表示用户使用的空间和文件数超过软限制后，过渡期的剩余时间。

由于 user1 已经超过软限制，因此如果继续向该文件系统放入文件，且放入后大于硬限制，将会显示以下内容：

```
#使用复制命令复制文件时将会得到提示信息，并终止操作
$ cp ~/linux2.iso linux2.iso
sdd1: write failed, user block limit reached.
cp: writing `linux2.iso': Disk quota exceeded
```

如果有多个文件系统设置了磁盘配额限制，要查看系统中所有磁盘的使用情况，可以使用 repquota 命令的选项 a。

```
#使用选项 a 查看所有磁盘配额的使用情况
# repquota -a
```

2．校准磁盘使用情况

磁盘配额使用一段时间后，系统记录的磁盘使用情况可能会出现偏差，这时可以采用手动校准的方式保持准确性。

手动校准磁盘的使用情况，需要使用前面介绍的 quotacheck 命令：

```
#使用 quotacheck 命令手动校准磁盘使用情况
# quotacheck -cug /dev/sdd1
```

7.6　文件系统维护基础

大多数时候，文件系统是不需要维护的，但一些特殊情况下需要管理员对文件系统进行维护，例如文件系统空间告警，电源故障、不正常关机导致数据出现错误，人为的误操作等。

文件系统维护能够大大减少系统故障，并保证重要数据和业务正常运行，因此初学者应该掌握维护文件系统的命令和使用方法。本节将简单介绍文件系统维护的命令、方法和思路。

7.6.1　查看文件系统使用情况命令 df

在产生数据较多的生产系统上（例如数据库服务器），管理员需要时刻保持警惕，以

避免因文件系统空间不足导致生产停止。通常管理员会使用 df 命令来查看文件系统的使用情况，本小节将简单介绍 df 命令的使用方法。

df 是一个非常简单的命令，它没有复杂的参数，也没有过多的选项，直接使用即可：

```
#使用 df 命令查看文件系统的使用情况
# df
Filesystem       1K-blocks      Used     Available      Use%    Mounted on
/dev/sda2         59547684   2263632      54210416        5%             /
/dev/sda3         20308052    860888      18398924        5%         /home
/dev/sda1           388693     15921        352693        5%         /boot
tmpfs               257720         0        257720        0%      /dev/shm
/dev/sdd1         82567188   4162136      74210884        6%     /mnt/sdd1
```

df 命令一共输出了 6 个字段，这 6 个字段及其含义如下。

- Filesystem：文件系统名称。
- Size 或 1K-blocks：文件系统空间大小（单位为 KB）。
- Used：已经使用的空间大小。
- Available：目前可用的空间大小。
- Use%：已使用的空间占总数的百分比。
- Mounted on：文件系统的挂载点。

可以通过查看 Use%字段的方式，查找空间不足的文件系统。其中要特别注意根分区和保存有重要生产数据的空间。如果可用空间不足，可以考虑扩展文件系统，也可以删除占用空间较大的文件。

如果觉得上面的命令输出看起来很不方便，可以使用选项 h，以更直观的方式查看：

```
#使用选项 h 以更直观的方式显示容量
# df -h
Filesystem       Size        Used    Avail       Use%     Mounted on
/dev/sda2        57G         2.2G    52G         5%       /
/dev/sda3        20G         841M    18G         5%       /home
……
```

提示：像数据库这种随时间增长，数据量也随之增长的应用系统，应该考虑将其数据文件保存在逻辑卷等可在线扩充的文件系统上。

7.6.2　追踪大文件命令 du

随着系统运行时间的持续，用户数据越来越多，可能会使文件系统的可用空间越来越小，直到有一天收到系统磁盘空间告警的消息。这时管理员可能需要查看是哪些文件占用了过多的空间，以清理文件系统上的无用文件。本小节将简单介绍如何使用 du 命令查找文件系统上的大文件。

du 命令的作用是显示目录中的文件列表，并显示文件大小。

【命令格式】

```
du [option] [directory]
```

如果不指定目录参数，命令 du 将显示当前目录中的文件及其大小。

【常用选项】

- ❑ h：以更直观的方式显示文件大小。
- ❑ s：查看当前目录的大小（而不是目录中的文件）。

【用法示例】

对于可用空间较小的文件系统，可以使用命令 du 追踪大文件及目录的具体位置。

（1）例如要追踪目录/mnt：

```
#使用 du 命令显示目录/mnt 中的文件大小
# du /mnt
……
3973920 /mnt/sdd1
8       /mnt/file
3973952 /mnt
```

（2）上面命令的输出看起来非常不方便，可以使用选项 h，以更加直观的方式查看文件大小：

```
#使用选项 h 以更直观的方式显示文件大小
# du -h /mnt
……
3.8G    /mnt/sdd1
8.0K    /mnt/file
3.8G    /mnt
```

（3）如果要查看当前目录的大小，可以配合 du 命令的选项 s 和 h：

```
#使用选项 s 显示目录的大小
# du -sh
47M     .
```

从上面几个命令的输出可以看出，占用空间较大的目录是/mnt/sdd1。此时可以继续使用 du 命令查找目录/mnt/sdd1 中的大文件，直到找到占用空间最多的无用文件为止。

7.6.3　修复文件系统命令 fsck

由于系统电源故障、人为操作等原因，可能会导致文件系统出现不一致等故障。大多数时候，这些故障都可以通过修复文件系统来解决。本小节将简单介绍修复文件系统命令 fsck 和处理文件系统故障的简单思路。

1．文件系统修复命令fsck

在之前的章节中已经介绍过文件系统修复命令 fsck，此命令的主要功能是扫描并尝试修复文件系统中的错误。

【命令格式】

```
fsck [options] filesystem
```

该命令也是一个系列命令，除此之外，还有 fsck.etx3、fsck.ext2 等。

【常用选项】

- ❑ A：扫描/etc/fstab 中列出的所有文件系统。
- ❑ a：自动修复文件系统中的错误，不询问用户。
- ❑ f：强制扫描文件系统。
- ❑ t：指定文件系统类型。不使用此选项时，fsck 将会自动检测。

除此之外，还有许多不常用的选项，读者可以通过阅读帮助文档了解这些选项。

【用法示例】

使用 fsck 命令修复文件系统时，如果该文件系统处于挂载状态，建议先将其卸载或以只读方式重新挂载，然后再运行 fsck 命令修复文件系统。

（1）以只读方式重新挂载并扫描文件系统：

```
#使用 mount 命令的 ro、remount 参数以只读方式重新挂载/dev/sdc1
# mount -o ro,remount /dev/sdc1
#使用 fsck 命令扫描/dev/sdc1
# fsck /dev/sdc1
```

使用上面的示例命令时，由于文件系统正处于挂载状态，因此命令会发出警告，并要求用户确认。

（2）有时需要强制扫描文件系统，以修复其中的错误：

```
#使用选项 f 强制扫描文件系统
# fsck -f /dev/sdc1
```

（3）使用 fsck 命令扫描文件系统时，可能会询问用户采取何种操作（例如发现文件连接数不一致时）。如果要让 fsck 自动修复，可以使用选项 a：

```
#使用选项 a 让 fsck 自动修复发现的错误，不询问用户
# fsck -a /dev/sdc1
```

注意：不要使用 fsck 命令对 NTFS 等 Windows 分区或不支持的文件系统执行修复，以免损坏文件系统。

2. 处理文件系统故障思路

文件系统总是会出现故障，可能是硬盘快要“寿终正寝”，也可能是电源故障，导致文件系统数据不一致等。初学者可能比较难处理此类错误，此处将简单讨论处理文件系统故障的思路。

（1）文件系统故障发生在系统运行时

当正在使用的文件系统发生故障时，内核会立即将情况写入日志，因此一般此类错误都是通过阅读日志发现的。

如果此类故障发生在生产系统中，确定故障前一般不要重启系统，以免根文件系统发生故障无法启动。此时可以尝试如下操作。

- ❑ 使用 df 命令查看文件系统的挂载点和使用情况，以确认数据损失可能会发生在何处。
- ❑ 尽量减少文件系统的写入操作。如果磁盘发生了致命错误，写入操作可能会加重

磁盘故障。

- 如果发生故障的文件系统中保存有重要数据，且系统中的其他磁盘有可供存放备份文件的磁盘空间，此时可以使用 dd 等命令，对故障文件系统进行备份。如果系统中没有存放备份数据的空间，可以使用 WinSCP 等远程工具将数据备份至其他位置。
- 备份并检查重要数据后，可以使用 fsck 命令检查文件系统，并尝试修复。

在生产系统中处理此类故障时，一定要先确保数据安全，否则一旦数据丢失，将可能导致重大的事故。

（2）光盘引导救援模式

如果系统启动时，无法使用 grub 引导加载程序，或已确认根文件系统发生故障，此时可以使用光盘引导，并进入救援模式修复文件系统，或重新安装 grub 引导加载程序。

使用光盘引导后，在其安装模式选择界面中输入 linux rescue，按下 Enter 键即可启动救援模式。在救援模式中，可以重新安装 grub 引导加载程序或修复文件系统。

（3）系统启动时进入救援模式

可能由于一些特殊的原因，系统启动时进入了救援模式，如图 7.5 所示。

```
Setting hostname localhost.localdomain:                    [  OK  ]
No devices found
Setting up Logical Volume Management:                      [  OK  ]
Checking filesystems
/: clean, 89703/15418656 files, 1041884/15418383 blocks
fsck.ext3: No such file or directory while trying to open /dev/sdb1
/dev/sdb1:
The superblock could not be read or does not describe a correct ext2
filesystem.  If the device is valid and it really contains an ext2
filesystem (and not swap or ufs or something else), then the superblock
is corrupt, and you might try running e2fsck with an alternate superblock:
    e2fsck -b 8193 <device>

/home: clean, 101/5242880 files, 208433/5241198 blocks
/boot: clean, 33/50200 files, 17639/200780 blocks
                                                           [FAILED]

*** An error occurred during the file system check.
*** Dropping you to a shell; the system will reboot
*** when you leave the shell.
*** Warning -- SELinux is active
*** Disabling security enforcement for system recovery.
*** Run 'setenforce 1' to reenable.
Give root password for maintenance
(or type Control-D to continue): _
```

图 7.5　救援模式

系统进入救援模式的原因很多，常见的原因是非正常关机，从而导致文件系统数据破坏，无法启动系统。另一个常见原因是自动挂载文件/etc/fstab 不正确，导致文件系统无法挂载。

如果因为系统非正常关机，导致数据破坏无法启动，用户输入 root 用户密码进入救援模式，然后使用 fsck 命令修复文件系统，并重新启动系统即可。

如果自动挂载文件/etc/fstab 不正确而导致进入救援模式，一般需要通过修改自动挂载文件的方式修复系统。输入 root 用户密码进入救援模式，然后修改/etc/fstab 文件，重启系统即可。

提示：救援模式使用只读方式挂载所有磁盘，因此在救援模式中修改文件内容时，需要先使用 mount 命令的 remount 和 rw 等参数，以可读写方式挂载文件系统。

7.7　小　　结

- 7.1 节主要介绍了 Linux 系统中的磁盘管理，通过示例介绍如何对磁盘进行分区、修改分区类型等。
- 7.2 节讲解了 Linux 系统中的文件系统。包括文件系统简介，常用的文件系统，在分区上创建文件系统，使用文件系统等内容。
- 7.3 节主要介绍了 Linux 系统中的 RAID 设备。简单介绍了 RAID 的分类和阵列等级等基础知识。以 RAID 5 为例介绍了如何创建一个软 RAID 设备，以及具备数据冗余功能的阵列数据同步和恢复等内容。
- 7.4 节主要介绍了 LVM 的应用。LVM 允许在线扩充文件系统容量，移出磁盘等操作，解决了服务器存储的现实问题。本节从 LVM 结构入手，介绍逻辑卷的基本概念，利用实例介绍如何创建并管理逻辑卷。LVM 的应用范围非常广，因此初学者需要格外注意。
- 7.5 节介绍了 Linux 系统中的磁盘配额管理，包括建立磁盘配额支持、建立磁盘配额和管理磁盘配额等内容。
- 7.6 节主要介绍了如何维护文件系统，包括使用 df 命令查看文件系统的使用情况、修复文件系统、救援模式等内容。

磁盘和文件系统是 Linux 系统存储的基础，学好磁盘和文件系统管理，可以在实际应用中减少许多不必要的麻烦。除此之外，初学者可以按需要选择学习 RAID、LVM 和磁盘配额等内容。

第 8 章　Linux 系统管理

在前面几章的学习中，仅仅讲解了 Linux 系统的一些特性、使用方法和外部应用等，并未真正地接触到 Linux 系统内部。本章将介绍 Linux 系统本身的管理，内容包括服务管理、作业管理、计划任务等。具体的应用有：关闭系统的某项功能、杀死未响应的进程、让系统在某一时刻自动运行任务等。这些内容有助于初学者更好地使用和管理系统，因此初学者应该了解本章的大部分内容。

本章主要涉及的知识点如下。

- ❑ 详细讲解 Linux 系统中的系统服务管理，包括查看系统服务、设置系统服务的自启动状态等。
- ❑ 在 Linux 系统中查看进程、进程树，管理进程等。
- ❑ 利用 cron 和 at 自动执行计划任务。
- ❑ Linux 系统中的日志记录程序及其配置文件介绍，如何使用工具执行日志轮循、监控系统日志等。

8.1　系统服务管理

在 Linux 系统中有许多系统服务，这些系统服务涉及 Linux 运行的各个方面。例如提供打印的服务 cups，为系统提供网络支持的服务 network，系统网络防火墙服务 iptables 等。

在日常管理中，需要对这些服务进行管理，管理内容包括查看系统服务、设置服务自启动状态、重新启动服务等。关于服务管理的应用有许多例子，例如优化系统时，关闭不需要的打印服务、蓝牙支持服务、IPv6 防火墙等。本节将讲解如何管理 Linux 系统中的服务。

小知识：服务是指操作系统中提供指定功能的程序或进程。服务也是一类应用程序，不同的是服务一般只在后台运行，且服务的对象一般是本地系统或网络中的主机和用户。

8.1.1　查看系统服务

Linux 系统中存在许多服务，按服务对象不同，通常可以分为本地系统服务和网络服务两类。本地系统服务主要用于支撑本地系统运行，例如打印、网络支持、防火墙服务、蓝牙支持服务等。网络服务通常用于对网络中的用户或主机提供服务，例如 Web 应用服务、代理服务、FTP、数据库服务等。

无论何种服务，在 Linux 系统中都可以设置为在一个或多个特定的运行级别中自动启

动，即当系统进入某个指定的运行级别时，会自动运行设置在该级别中的服务。管理这些服务的第 1 步是查看系统中的服务，本小节将简单介绍如何查看系统服务。

查看 Linux 系统中的服务列表，可以使用命令 chkconfig。该命令可用于查看服务的自启动状态，也可用于设置服务的启动状态。

【命令格式】

```
chkconfig --list [name]
```

使用 chkconfig 命令的长格式选项 list，即可查看系统中的服务。

【用法示例】

chkconfig 命令可以查询单一服务的自启动状态，也可以用于查询系统中的服务列表。

（1）查询系统防火墙服务 iptables 的自启动状态：

```
#使用选项 list 加服务名称，查看指定服务的自启动状态
# chkconfig --list iptables
#服务名称后面的列表，表示服务在各级别中的启动状态
#on 表示启动，off 表示关闭
iptables        0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

从示例命令的输出中可以看到，系统防火墙在运行级别 2, 3, 4, 5 中会自动运行。

注意：使用选项 list 和服务名称查看服务时，必须使用服务的全名，否则将无法查询到结果。

（2）如果不使用服务名称参数，命令将会列出系统中的所有服务：

```
#使用选项 list 查看系统中所有服务的自启动状态
# chkconfig --list
NetworkManager  0:off   1:off   2:off   3:off       4:off   5:off   6:off
acpid           0:off   1:off   2:on    3:on        4:on    5:on    6:off
anacron         0:off   1:off   2:on    3:on        4:on    5:on    6:off
apmd            0:off   1:off   2:on    3:on        4:on    5:on    6:off
……
#以下为非独立服务的启动状态
xinetd based services:
       chargen-dgram:   off
       chargen-stream:  off
       daytime-dgram:   off
……
```

在上面的命令输出中，前面部分显示了服务在各运行级别中的自启动状态，最后几行仅仅输出了服务的运行状态，并没有输出服务的启动状态。

说明：像上面这个命令中输出的那样，在每个运行级别中都可以设置自启动状态的服务称为独立服务。

命令输出的最后几个服务是一些老式的服务集，今天已经很少有人使用这些服务。这些老式服务集的运行状态，依赖于一个名为 xinetd 的服务。通常将这些依赖于 xinetd 的服务称为非独立服务。非独立服务受 xinet 服务管理，因此如果要启动这些服务，应该首先启动 xinetd。

（3）有时可能会忘记服务的具体名称，也可能只记得服务名称中的一部分，这时可以将 chkconfig 命令与 grep 命令一起配合使用。

例如要查询服务名称中含有 nfs 的服务列表：

```
#先使用 chkconfig 命令列出所有服务，然后使用 grep 命令筛选含有 nfs 的行
# chkconfig --list | grep nfs
nfs             0:off   1:off     2:off   3:off   4:off   5:off   6:off
nfslock         0:off   1:off     :off    3:on    4:on    5:on    6:off
```

命令 grep 会在 chkconfig 命令的输出结果中查找，如果找到输出中含有 nfs 的行，就输出整行。

8.1.2　设置服务自启动状态

许多时候管理员都需要设置服务的自启动状态，关闭一些不需要的服务，或将需要的服务设置为在相应的运行级别启动等。

一个比较典型的例子：当系统供个人用户使用时，需要禁用网络服务，并且按个人需求禁用不必要的服务，以节省资源（例如没有打印机时，禁用打印服务）。本小节将简单介绍如何设置服务的自启动状态。

【命令格式】

```
chkconfig [--level <levels>] <name> <on|off>
```

在上面的命令格式中，尖括号“<”、“>”中的内容表示必须使用的参数。如果使用了选项 level，必须为参数指定运行级别列表，并且设置自启动状态必须指定服务名称和启动状态。

【用法示例】

使用命令 chkconfig 既可以设置独立服务的自启动状态，也可以设置非独立服务的自启动状态。

（1）设置独立服务系统防火墙的自启动状态为关闭：

```
#使用 chkconfig 命令设置防火墙服务的自启动状态为关闭
# chkconfig iptables off
#使用选项 list 查看并验证结果
# chkconfig --list iptables
iptables        0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

这时系统防火墙在所有运行级别下都为关闭状态。

（2）如果要设置开启防火墙，可以用如下命令：

```
#使用 chkconfig 命令设置防火墙的自启动状态为开启，并验证设置
# chkconfig iptables on
# chkconfig --list iptables
iptables        0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

这时防火墙将会在运行级别为 2, 3, 4, 5 时自动启动。

（3）从上面几个示例命令的结果中可以看出，设置服务为开启状态时，命令默认将其设置为运行级别 2, 3, 4, 5 为开启。设置服务关闭时，默认设置为所有运行级别关闭。显然

这个默认设置并不适用于所有用户和服务，这时可以使用选项 level 指定运行级别列表。

例如设置系统防火墙在运行级别 3, 5 中自动启动：

```
#使用选项 level 设置自启动运行级别列表并验证
# chkconfig --level 35 iptables on
# chkconfig --list iptables
iptables        0:off   1:off   2:off   3:on    4:off   5:on    6:off
```

上面的命令中，使用了长格式选项 level 指定运行级别列表，使用这种方式指定服务要自动启动的多个运行级别列表。

（4）对于非独立服务而言，由于没有运行级别这个概念，因此不能使用 level 选项指定运行级别列表，可以直接设置这些服务的状态。

例如要将简单文件传输协议（TFTP）服务设置为开机自启动：

```
#使用 chkconfig 命令设置非独立服务的启动状态并验证
# chkconfig tftp on
# chkconfig --list tftp
tftp           on
```

由于非独立服务受服务 xinetd 的管理，因此还应该将 xinetd 设置为自动启动：

```
#使用 chkconfig 命令设置 xinetd 服务的启动状态
# chkconfig --level 345 xinetd on
# chkconfig --list xinetd
xinetd          0:off   1:off   2:off   3:on    4:on    5:on    6:off
```

注意：虽然可以使用 chkconfig 命令设置非独立服务的自启动状态，但这并不能让所有非独立服务启动。有些非独立服务可能还需要手动修改其配置文件，这样 xinetd 服务启动后才能启动这些非独立服务。

（5）Red Hat 的发行版中，通常还可以使用工具 ntsysv 设置服务的自启动状态。在字符界面中执行命令 ntsysv 即可启动该工具，其运行界面如图 8.1 所示。

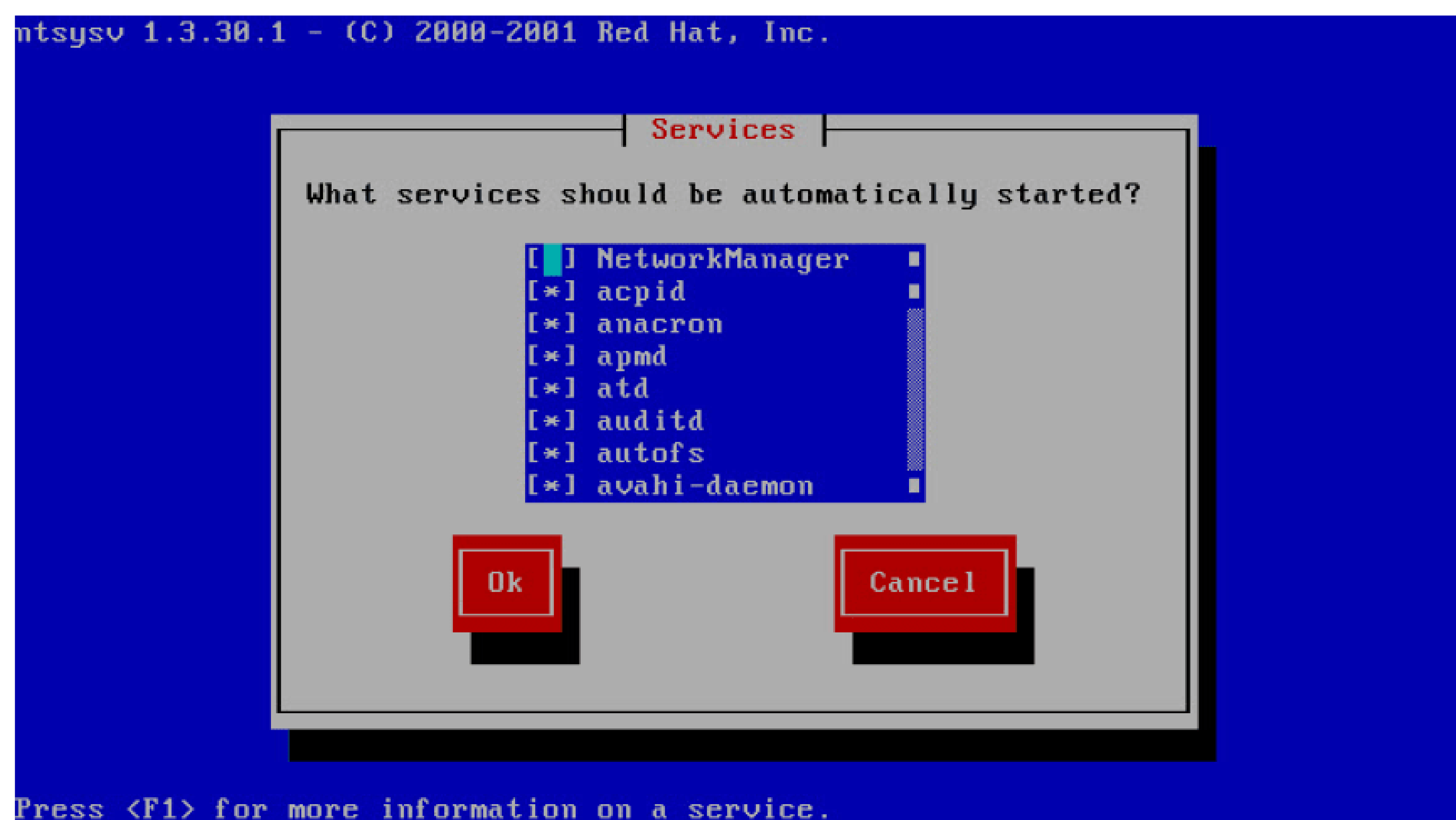


图 8.1　ntsysv 工具

使用 ntsysv 工具时，按上下方向键找到要设置的服务名称。将光标停留在相应的服务

名称前面的“[　]”内，按空格键即可选中该项。要确认选择时，使用 Tab 键切换到 OK 按钮，按 Enter 键或空格键即可完成设置。

8.1.3　添加删除系统服务

chkconfig 命令不仅可以查看、设置服务的自启动状态，还可以添加系统服务（前提条件是有可供使用的运行级别脚本）。本小节将简单介绍如何添加系统服务。

【命令格式】

```
chkconfig -add <name>
chkconfig -del <name>
```

使用 chkconfig 命令的 add 和 del 选项并提供系统服务名称即可添加、删除系统服务。

【用法示例】

要添加系统服务，首先需要运行级别脚本（在本书的第 19 章中将介绍如何编写运行级别脚本）。此处将以 MySQL 5.1.55 源码包中提供的运行级别脚本为例，演示如何添加、删除系统服务。

（1）MySQL 5.1.55 编译安装完成后，可以复制源码目录中的 support-files/mysql.server 至系统服务脚本目录/etc/init.d：

```
#使用 cp 命令复制 MySQL 的运行级别脚本至系统服务目录，并重命名为 mysqld
# cp support-files/mysql.server /etc/init.d/mysqld
#由系统运行级别脚本是一个可执行文件，因此还需要为其添加可执行权限
# chmod +x /etc/init.d/mysqld
```

完成此步操作后，chkconfig 命令就能自动检测到新加入的运行级别脚本了。

说明：编译安装是 Linux 等类 UNIX 系统中常见的软件安装方式之一，将在本书的第 9 章中介绍。

（2）此时可以使用 chkconfig 命令将 MySQL 的运行级别脚本添加为系统服务：

```
#使用 chkconfig 命令添加系统服务，并验证结果
# chkconfig --add mysqld
# chkconfig --list mysqld
mysqld          0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

从上面的命令输出中可以看出，mysqld 服务已经成功添加。chkconfig 命令还自动为其设置了自启动状态。

（3）从系统中移除 MySQL 后，还需要将其系统服务也一并移除，以免产生不必要的错误：

```
#使用命令 chkconfig 删除系统服务 mysqld
# chkconfig -del mysqld
#使用 chkconfig 命令查看时，提示用户应该使用选项 add 添加
# chkconfig -list mysqld
service mysqld supports chkconfig, but is not referenced in any runlevel
(run 'chkconfig --add mysqld')
```

由于 chkconfig 命令仅将 mysqld 服务移除，并未删除其脚本，因此如果确认不再需要使用服务的运行级别脚本，可以使用 rm 命令将其运行级别脚本删除。

本小节仅以 MySQL 的运行级别脚本为例，介绍如何添加、删除系统服务。如果需要为其他软件添加系统服务，可以参考软件中的 README 等说明文件。

注意：并不是所有发行版都支持 chkconfig 命令（例如 Gentoo 使用 rc-update 管理系统中的服务），因此在某些发行版中可能需要阅读说明或帮助文档。

8.1.4　启动和关闭服务命令 service

许多时候需要在系统处于运行状态时启动和关闭某个服务。最常见的例子是管理员在配置一个服务时，需要反复关闭、启动服务（启动时服务将重新读取配置文件），以验证配置文件是否能够达到预期的目的。也可能管理员正在对系统做某项测试，需要关闭某些服务，以消除这些服务对测试结果的影响等。例如测试网络服务时，一般都需要关闭防火墙服务。

像上面这些情况都需要手动关闭、启动服务。目前许多 Linux 系统都支持使用命令 service 启动和关闭服务。本小节将简单介绍 service 命令的用法。

【命令格式】

```
service name [start|stop|restart]
```

使用 service 命令时，必须要指明准确的服务名称及服务执行的动作。服务执行的动作中，start 和 stop 分别表示启动、关闭服务，restart 表示重新启动服务。有些服务也可以使用 save（保存）和 reload（重新载入配置文件）等动作。

【用法示例】

（1）例如要关闭系统防火墙 iptables：

```
#使用 service 命令关闭系统防火墙
# service iptables stop
#由于系统防火墙以模块的形式加载到系统中（不同于其他服务），因此关闭时，也卸载了相关模块
Flushing firewall rules:                                [  OK  ]
Setting chains to policy ACCEPT: filter                 [  OK  ]
Unloading iptables modules:                             [  OK  ]
```

（2）有时修改了服务的配置文件，希望以新的配置文件运行服务，这时可以使用 restart 参数让服务重新启动，以重新读取配置文件。

例如管理员通过修改配置文件的方法重新配置了网络接口卡的 IP 地址等信息，并希望系统网络重新启动：

```
#使用 service 重新启动系统网络支持服务 network
# service network restart
Shutting down interface eth0:                           [  OK  ]
Shutting down loopback interface:                       [  OK  ]
Bringing up loopback interface:                         [  OK  ]
Bringing up interface eth0:                             [  OK  ]
```

执行以上命令后，系统会先关闭网络支持服务 network，然后再读取配置文件，并使

用新的配置文件启动网络接口。

（3）然而有些发行版不支持 service 命令，由于服务的启动脚本都放在目录/etc/init.d中，因此可以使用手动运行的方式执行启动脚本。

例如要启动系统计划任务服务 crond：

```
#使用全路径方式启动服务
# /etc/init.d/crond start
Starting crond:                                            [  OK  ]
```

（4）许多服务除了启动、停止和重启之外，还有一些其他动作，如保存、重新载入配置文件等（并非所有服务都支持这些动作）。

例如要保存系统防火墙规则：

```
#使用 save 动作保存防火墙规则
# service iptables save
Saving firewall rules to /etc/sysconfig/iptables:   [  OK  ]
```

注意：Linux 系统中有许多服务，发行版不同服务也有所差异，因此要了解这些服务的具体功能，需要阅读发行版的相关说明。

8.2　进程管理命令

许多 Linux 系统运行于多用户环境下，管理员有必要管理主机的用户进程，以避免消耗过多的系统资源。也可能出于其他的原因需要管理进程，例如强制杀死一个未响应的进程等。本节将简单介绍在 Linux 系统中如何查看、管理进程、设置进程优先级等内容。

小知识：Linux 系统启动时，运行的第 1 个进程名为 init。这个进程是所有进程的父进程，其 PID 永远是 1。PID 是系统分配给进程的唯一标识符，虽然系统中可能存在两个名称一样的进程，但 PID 绝对不同。当系统启动进程时，init 会通过系统调用的方式启动新进程，并为新进程分配资源和 PID 等。而当进程结束时，系统会收回分配给进程的资源和 PID 等，这些资源和 PID 可以重复利用，下一次系统可能会将这些资源又分配给另一个进程。

8.2.1　查看进程命令 ps

要查看系统中运行的进程，可以使用 ps 命令，这是查看系统进程最常用的一个命令。ps 命令会输出系统中的进程详细信息，管理员可以利用这些信息杀死进程，也可以管理系统中的进程。

【命令格式】

```
ps [option]
```

【常用选项】

- a：显示所有终端进程。

- u：显示所有用户的进程。
- x：显示所有进程，包括没有明确终端的进程。
- e：显示所有进程。此选项与选项 x 类似，但使用的格式为 BSD 系统格式。
- f：显示 UID，PPID（父进程 ID），C（CPU 占用率）和 STIME（进程启动时间）字段。
- l：以长格式显示进程列表。

【用法示例】

（1）如果不加任何参数，ps 命令将会输出当前用户的进程：

```
#以普通用户身份运行 ps 命令
$ ps
  PID TTY          TIME       CMD
12666 pts/1     00:00:00    bash
12688 pts/1     00:00:00    ps
```

上面是以普通用户的身份运行的 ps 命令，命令输出了当前用户使用的两个进程。其中包括了进程的 PID、使用的终端、占用 CPU 的时间和启动进程的命令等内容。

（2）许多时候管理员会执行这个命令，以查看系统中运行的进程，做出相应的管理动作，例如查看系统上哪些进程过多地使用了 CPU、内存等资源。

要显示用户自身进程的详细情况，可以配合使用选项 u：

```
#使用选项 u 查看用户自身进程的详细情况
# ps -u
USER  PID   %CPU %MEM VSZ   RSS  TTY   STAT  START TIME COMMAND
root  3606  0.0  0.0  1656  428  tty2  Ss+   Oct08 0:00 /sbin/mingetty
root  3608  0.0  0.0  1656  432  tty3  Ss+   Oct08 0:00 /sbin/mingetty
……
root  12720 0.0  0.1  4252  940  pts/0 R+    10:10 0:00 ps u
```

上面的命令输出了系统正在运行进程的详细信息。命令输出的各字段含义如下。

- USER：运行此进程的用户名称。与 Windows 不同，系统进程的用户是 root（Windows 是 system 用户）。
- %CPU：进程的 CPU 占用率。CPU 占用率越高的进程，使用的计算资源越多。
- %MEM：实际内存占用率。
- VSZ：进程占用虚拟内存大小，单位是 KB。
- RSS：占用物理内存大小，单位是 KB。
- STAT：目前进程所处的状态属性。
- START：进程开始运行的时间。
- TIME：进程占用 CPU 的时间总和。
- COMMAND：启动进程使用的命令。

从上面的命令输出中可以看到当前用户的进程占用系统资源的情况。

在命令输出中有一个比较特殊的字段 STAT，一般由 1～3 个符号组成，它描述了进程当前的状态属性。常见的进程状态标识及所表达的含义如表 8.1 所示。

表 8.1　进程状态标识及其含义

进程状态标识	进程状态标识的含义
D	不可中断的等待状态，通常是等待 I/O 设备（磁盘及网络等）的数据等
R	正处于运行列队中的进程
S	正处于中断休眠状态的进程，该进程可能是在等待某个中断消息
T	已停止工作的进程，因其被跟踪所以存在
X	已经死亡的进程，通常不会看到这类进程
Z	已经僵死的进程
<	高优先级进程
N	低优先级进程
s	会话的管理者
+	进程会使用前台的终端
l	这是一个多线程的进程

了解以上状态的表达字符非常有助于管理员了解这些进程当前的状态，以便于更好地管理这些进程。

提示： 在 Linux 系统中，僵死的进程是指已经退出并正在释放系统资源的进程，并非未响应的进程。

（3）大多数情况下可能会希望看到所有用户及系统的进程，此时可以使用选项 a 和 x。这两个选项的含义分别是显示所有进程，以及没有明确终端的进程（通常是一些后台服务进程）。

查看当前系统中所有进程：

```
#使用选项 aux 查看当前系统中的所有进程
# ps -aux
USER  PID   %CPU  %MEM  VSZ   RSS   TTY    STAT  START  TIME  COMMAND
root  1     0.0   0.1   2064  620   ?      Ss    Oct08  0:01  init [3]
root  2     0.0   0.0   0     0     ?      S<    Oct08  0:00 [migration/0]
root  3     0.0   0.0   0     0     ?      SN    Oct08  0:00 [ksoftirqd/0]
root  4     0.0   0.0   0     0     ?      S<    Oct08  0:00 [watchdog/0]
……
wlh   12666 0.0   0.2   4532  1428  pts/1  Ss+   09:53  0:00   -bash
root  13343 0.0   0.1   4252  936   pts/0  R+    15:55  0:00   ps aux
```

从上面的输出中可以看到许多进程都没有终端，这些进程大多与系统服务相关，并不需要使用终端。

（4）ps 命令的另一个重要用途是检测服务是否成功启动。例如使用 mysqld_safe 手动启动了数据库，需要检查其运行状态：

```
#使用 ps 命令检测 mysql 服务的运行状态
#由于 ps 命令会输出 grep 进程，因此使用 grep 命令的选项 v 反转显示
# ps -aux | grep mysqld | grep -v grep
root      3529  0.1  0.4   4492  1128 pts/0    S    17:29   0:00 /bin/sh
/usr/local/mysql/bin/mysqld safe --user=mysql
mysql      3655   0.4  13.0  297488  33416  pts/0     Sl    17:29     0:00
```

```
/usr/local/mysql/libexec/mysqld --basedir=/usr/local/mysql --datadir=/
data/mysql db --user=mysql --log-error=/data/mysql db/localhost.
localdomain.err --pid-file=/data/mysql db/localhost.localdomain.pid --socket=
/tmp/mysql.sock --port=3306
```

在上面的结果中，第 1 个 PID 为 3529 的进程，这是 root 用户启动 mysql 服务时使用的命令。第 2 行是 PID 为 3655 的进程，这就是以 MySQL 的专有用户 mysql 启动的 mysql 服务。

关于 ps 命令的选项 e、f 和 l，将在进程优先级的相关内容中介绍。

ps 是一个非常复杂的命令，有许多选项、用法，在不同的系统中，显示的内容也会有差异，因此在其他系统中使用时，需要阅读帮助文档或有关说明。

注意：对于许多初学者而言，可能对本节中的进程等相关内容十分陌生，读者可以阅读相关书籍了解系统、进程等运行的方式及相关概念。

8.2.2　进程树

在 Linux 系统中，父进程、子进程之间的关系非常复杂。例如一个进程可以有一个或多个子进程，而子进程也可以再衍生出新的子进程。

Linux 系统使用了一棵树的方法表示进程与子进程之间的关系，这种方法非常简洁地表示出进程间的父子关系。要查看 Linux 系统中的进程树，可以使用命令 pstree。

例如，查看当前系统中的进程树：

```
#使用 pstree 命令查看当前系统中的进程树
# pstree
init─┬─acpid
     ├─atd
     ├─auditd─┬─audispd───{audispd}
     │        └─{auditd}
     ├─automount───4*[{automount}]
     ├─avahi-daemon───avahi-daemon
     ├─crond
     ├─cupsd
     ├─dbus-daemon───{dbus-daemon}
……
```

从上面的进程树可以非常清楚地看出，INIT 进程是所有进程的父进程，负责管理系统中所有的进程。

在本示例中，INIT 进程的子进程 automount 衍生出一个新的子进程 4*[{automount}]，其中的“4*”表示当前进程有 4 个线程。有时需要使用这种方式验证网络服务的线程数，因此应该特别注意。

8.2.3　实时显示进程命令 top

前两个小节中讲解到的查看命令查看的系统进程都是静态的，不能实时地查看进程占用系统资源的情况。如果要实时显示系统中的进程，可以使用 top 命令。

top 命令不仅可以实时地查看进程占用系统资源的情况，还能够按使用系统资源进行排序，它是目前 UNIX/Linux 等系统中最为流行的系统性能分析工具。本小节将简单介绍实时显示进程命令 top。

（1）使用 top 命令时，无须使用任何参数和选项：

```
#使用 top 命令实时显示进程
# top
#以下为交互式显示界面
top - 17:10:03 up 1 day, 20:28,  3 users,  load average: 0.00, 0.00, 0.00
Tasks:  84 total,   3 running,  81 sleeping,   0 stopped,   0 zombie
Cpu(s): 0.0%us,   0.7%sy,   0.0%ni,  99.0%id,   0.0%wa,   0.0%hi,   0.3%si,
0.0%st
Mem:    515444k total,  358736k used,   156708k free,    61444k buffers
Swap:   1044184k total, 0k used,  1044184k free,   219528k cached

  PID USER   PR  NI   VIRT  RES   SHR  S %CPU %MEM   TIME+  COMMAND
 3545 root   18  0    1968  644  564   S  0.3  0.1   0:46.35 hald-addon-stor
    1 root   15  0    2064  620  532   S  0.0  0.1   0:01.90   init
    2 root   RT  -5      0    0    0   S  0.0  0.0   0:00.00   migration/0
    3 root   34  19      0    0    0   S  0.0  0.0   0:00.00   ksoftirqd/0
    ……
  244 root   16  -5      0    0    0   S  0.0  0.0   0:00.00   aio/0
```

top 命令显示了一个交互式的页面，命令会实时地更新这个页面，显示进程的 PID、用户、CPU 占用率等。

top 页面的前半部分显示了当前系统的统计信息，第 1 行显示了 top 启动的时间，期间登录的用户及负载均衡平均值。3 个负载均衡平均值分别显示过去的 1 分钟、5 分钟和 10 分钟内系统的负载均衡平均值。

小知识： 系统负载平均值（也叫负载均衡值）是一个很独特的概念，Linux 使用数字标识系统的负载，数字越大表明系统负载越高。在单处理器系统上，如果负载均衡值等于 1，表示系统处于满负荷状态，大于 1 表示系统处于超负荷状态。在多处理器或者多核处理器系统中，满负荷运行状态的负载均衡值应该与 CPU 数量或处理器核数相等。

第 2 行是系统中的进程统计信息，分别是进程总数、正在运行的进程数、处于睡眠状态的进程数、停止进程数及僵死进程数。

第 3 行是 CPU 的使用情况，这些 CPU 百分比分别是用户进程占用百分比、系统进程占用百分比、用户改变过优先级的进程占用百分比，以及空闲百分比等。

第 4 行和第 5 行分别是内存和交换空间使用情况，包括内存总量、已用内存、空闲内存等。

top 命令前半部分的最后 3 行是系统资源占用及分配情况的具体表现，在使用 top 命令时应该特别注意。

top 命令的后半部分是系统中的进程详细情况，许多字段在前面几个小节的命令中已经介绍过。除此之外 top 命令还提供了几个新的字段，这几个新字段及其含义如下。

- PR：进程的优先级。

- NI：该进程的优先级值。
- VIRT：进程使用虚拟内存总量。
- RES：进程使用的物理内存总量。
- SHR：进程使用的共享内存大小。

使用 top 命令后，可以按 Q 键退出 top 交互页面。

（2）通常管理员可能最为关心的是进程的 CPU 占用率及占用内存的大小等因素，因为这些因素对系统整体性能的影响最大。这时可以使用排序的方法查看进程列表。要按 CPU 使用率排序，可以直接按 P 键，top 将按 CPU 使用率由高到低排序：

```
#在 top 运行时按 P 键
top - 20:48:31 up 2 days, 6 min,  3 users,  load average: 0.00, 0.00, 0.00
Tasks:  88 total,   2 running,  86 sleeping,   0 stopped,   0 zombie
Cpu(s): 0.0%us, 0.3%sy, 0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    515444k total,   374484k used,   140960k free,    75908k buffers
Swap:  1044184k total,        0k used,  1044184k free,   219536k cached

  PID USER     PR  NI  VIRT  RES   SHR  S %CPU %MEM    TIME+     COMMAND
13474 root     15   0  2196 1004  808  R  0.3   0.2   0:38.44          top
    1 root     15   0  2064  620  532  S  0.0   0.1   0:01.91         init
    2 root     RT  -5     0    0    0  S  0.0   0.0   0:00.00  migration/0
    3 root     34  19     0    0    0  S  0.0   0.0   0:00.00  ksoftirqd/0
    4 root     RT  -5     0    0    0  S  0.0   0.0   0:00.02   watchdog/0
……
```

（3）如果需要按内存占用率排序，可以使用 M 键：

```
#在 top 运行时按 M 键
top - 20:56:48 up 2 days, 15 min,  3 users,  load average: 0.00, 0.00, 0.00
Tasks:  88 total,   1 running,  87 sleeping,   0 stopped,   0 zombie
Cpu(s): 0.3%us,  0.0%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi, 0.0%si, 0.0%st

  PID USER    PR  NI  VIRT RES   SHR   S %CPU   %MEM    TIME+     COMMAND
13878 root    15   0  2196 1000  804   R  0.7    0.2   0:00.05         top
    1 root    15   0  2064  620  532   S  0.0    0.1   0:01.91        init
    2 root    RT  -5     0    0    0   S  0.0    0.0   0:00.00  migration/0
    3 root    34  19     0    0    0   S  0.0    0.0   0:00.00  ksoftirqd/0
    4 root    RT  -5     0    0    0   S  0.0    0.0   0:00.02   watchdog/0
……
```

（4）top 命令还有许多使用方法，可以使用 H 键查看其帮助信息：

```
#top 运行时按 H 键
Help for Interactive Commands - procps version 3.2.7
Window 1:Def: Cumulative mode Off.  System: Delay 3.0 secs; Secure mode Off.

  Z,B       Global: 'Z' change color mappings; 'B' disable/enable bold
  l,t,m     Toggle Summaries: 'l' load avg; 't' task/cpu stats; 'm' mem info
  1,I       Toggle SMP view: '1' single/separate states; 'I' Irix/Solaris mode

  f,o        . Fields/Columns: 'f' add or remove; 'o' change display order
```

```
  F or O      . Select sort field
  <,>         . Move sort field: '<' next col left; '>' next col right
  R,H         . Toggle: 'R' normal/reverse sort; 'H' show threads
……
```

在帮助模式下按任意键，即可返回 top 命令的主界面。

top 是 Linux 系统中主要的性能监视工具之一，许多时候管理员都要使用到这个工具。关于这个工具的更多用法，读者可以查阅相关文档及 man 手册页。

8.2.4　将任务放在后台执行

执行一个花费时间很长的命令时，命令会长时间占用终端前台，这对于用户而言可能会非常不方便。例如将一个巨大的文件或目录移动到新的位置、从一个很庞大的文件系统上查找某个文件等，执行这些任务都十分耗时。这时可能希望将这些命令放在后台执行，与此同时用户还可以使用终端完成其他的任务。本小节将简单介绍如何将任务放在后台执行。

本节将使用 sleep 命令作为示例，该命令产生的进程会在指定时间内一直处于睡眠状态，直到时间结束，才退出并释放占用系统资源。如果在前台执行，此命令将会锁住终端直到时间结束。

在执行命令时，可以在命令最后加入符号&（读音与单词 and 相同），此时系统会将命令或程序放入后台执行。

（1）例如要创建一个 sleep 进程，持续时间为 300 秒，并将其切换至后台运行：

```
#使用&将命令放在后台执行
# sleep 300 &
[1] 16997
#使用 ps 命令查看后台进程
# ps
  PID TTY          TIME CMD
16568 pts/0    00:00:00 bash
16997 pts/0    00:00:00 sleep
16998 pts/0    00:00:00 ps
```

上面这个示例中，命令 sleep 开始执行即会立即切换至后台，使用 ps 命令即可发现 sleep 进程正在后台执行。

（2）使用&将命令放在后台执行时，处于后台的命令会直接将输出和错误信息输出到控制台。这样非常不方便查看，此时建议使用重定向将命令输出、错误信息输出到文件，以便于事后查看。

例如使用 mv 命令移动数据时，将错误重定向：

```
#使用重定向功能将命令的输出和错误信息写入文件 message 中
# mv /mnt/sdd5/test /root/ &>message &
```

在上面的示例中，使用重定向将命令产生的输出和错误信息写入文件 message 中，这样命令执行过程中就不会有消息发送到控制台。

8.2.5　查看后台任务命令 jobs

在上一小节中，介绍了如何将命令放在系统后台执行。但有时要将放在后台执行的命

令重新放到前台。将命令放到前台之前，需要知道命令的作业号。查看后台任务的作业号，可以使用命令 jobs。

要查看处于后台任务的作业号：

```
#使用 jobs 命令查看后台执行的命令
# jobs
[1]+  Running                 sleep 300 &
```

从命令的输出可以看到，有一个处于后台的任务，任务的作业号为 1，该任务的状态为正在运行。

使用 jobs 命令时需要注意，该命令显示的后台任务是用户手动执行的，需要占用前台的命令，不包含系统运行的后台进程。

8.2.6　后台任务调至前台命令 fg

许多时候需要将原本处于后台执行的任务重新放到前台，可能的情况是用户已经完成其他任务，也可能是用户想查看任务的进度等。这时可以使用 fg 命令将后台任务重新调至前台执行。

使用 fg 命令调取任务时，需要使用任务的作业号作为参数。例如将指定的作业重新放到前台执行：

```
#使用&符号将任务放至后台执行
# sleep 60 &
[1] 17059
#使用 jobs 命令查看后台任务的作业号
# jobs
[1]+  Running                 sleep 60 &
#使用 fg 命令将作业号为 1 的任务放到前台执行
# fg 1
sleep 60
```

上面的示例命令中，先使用 sleep 命令创建作业并放入后台，然后使用 jobs 命令查看后台的作业，最后使用命令 fg 将作业号为 1 的任务放到前台执行。

8.2.7　终止进程命令 kill

许多时候需要强行终止一个进程，可能的情况是不再需要继续运行这个进程，也可能是这个进程没有响应需要终止等。在 Linux 系统中，终止进程可以使用命令 kill。该命令执行时，会试图给进程发送信号以终止进程。

【命令格式】

```
kill -signal PID
```

在上面的命令格式中，signal 表示要发送给进程的信号，PID 表示进程的 ID 号（可以使用 ps 命令查看）。

在 Linux 系统中，可供 kill 命令使用的信号非常多。可以使用选项 l 查看这些信号：

```
#使用 kill 命令的选项 l 查看信号列表
# kill -l
 1) SIGHUP         2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP        6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL        10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE        14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD        18) SIGCONT     19) SIGSTOP     20) SIGTSTP
……
```

命令列出了可以使用的信号及其编号。在实际执行命令时，可以使用信号的名称，也可以使用信号编号。对于用户而言，命令中列举的大部分信号都不会用到，本书中仅介绍一些比较常用的信号。

【用法示例】

（1）大多数时候，并不需要发送特别的信号，而是以正常方式杀死进程。以正常方式杀死进程时，kill 命令将使用信号 15，即 SIGTERM（使用软件终止的方式）。这种方式会存在一些问题：进程的子进程可能会无法终止，并继续占用系统资源。

此处以命令 sleep 作为示例：

```
#使用 ps 命令查看终止命令的 PID 号
# ps
  PID TTY          TIME CMD
16568 pts/0    00:00:00 bash
16888 pts/0    00:00:00 sleep
16889 pts/0    00:00:00 ps
#使用 kill 命令终止 PID 为 16888 的进程
# kill 16888
```

上面的命令示例中，先使用 ps 命令查看要杀死进程的 PID，然后再使用 kill 命令将其杀死。

（2）如果管理员发现某个进程无法终止，可能的原因是进程已经无法响应。这时可以使用信号 9，强制终止无法终止的进程：

```
#使用 kill 命令强制终止 PID 为 16952 的进程
# kill -9 16952
#使用 ps 命令查看结果
# ps
  PID TTY          TIME CMD
16568 pts/0    00:00:00 bash
16956 pts/0    00:00:00 ps
[1]+  Killed                  sleep 400
```

从 ps 命令的输出中可以看到，使用信号 9 强制终止进程后，进程会立即被杀死，系统正在回收资源。

注意：强制终止进程并不能杀死所有进程，有时使用强制终止之后，进程仍然没有结束，这时可能需要重新启动系统。

（3）有时需要让一个进程停止执行，这时可以使用信号 19 挂起进程：

```
#使用 ps 命令查看进程的 PID
```

```
# ps
  PID TTY          TIME CMD
16568 pts/0    00:00:00 bash
16924 pts/0    00:00:00 sleep
16931 pts/0    00:00:00 ps
#使用信号 19 挂起 PID 为 16924 的进程
# kill -19 16924
```

执行上面的命令之后，进程将会暂时停止执行，可以使用 top 等命令查看其状态。

提示：使用信号 19 挂起进程后，如果不结束、继续执行挂起的进程，它将会一直存在于系统中，直到系统关闭。

（4）如果要让一个已经停止的进程重新运行，可以向其发送信号 18：

```
#向 PID 为 16924 的进程发送信号 18
# kill -18 16924
#使用 ps 命令查看进程状态
# ps
  PID TTY          TIME CMD
16568 pts/0    00:00:00 bash
16946 pts/0    00:00:00 ps
[1]+  Done                    sleep 30
```

从上面的输出中可以看到，进程已经重新开始执行。

（5）有时正在执行一个很花费时间的任务，可能事先没有估计任务完成的时间。由于正在运行的任务锁住了终端，用户将无法使用 kill 命令，此时可以使用快捷键 Ctrl+C 退出任务。

例如运行 ping 命令时，使用快捷键 Ctrl+C 使进程退出：

```
#使用 ping 命令测试与目标主机间的连通情况
# ping 192.168.44.1
PING 192.168.44.1 (192.168.44.1) 56(84) bytes of data.
64 bytes from 192.168.44.1: icmp seq=1 ttl=255 time=10.7 ms
64 bytes from 192.168.44.1: icmp seq=2 ttl=255 time=4.09 ms

#此时按下快捷键 Ctrl+C
--- 192.168.44.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 4.092/7.437/10.782/3.345 ms
```

当用户按下快捷键 Ctrl+C 后，系统会向该进程发送一个中断信号，迫使其退出。许多命令都支持快捷键 Ctrl+C 退出，因此这个快捷键十分有用。

（6）对于正在前台运行的比较特殊的进程，可能需要使用更为强制的手段，这时可以考虑使用快捷键 Ctrl+Z。快捷键 Ctrl+Z 与信号 19 一样，会强制将前台进程放入后台，并停止运行。

例如使用快捷键 Ctrl+Z 强制停止正在前台运行的 sleep 进程：

```
#使用 sleep 命令，并使用快捷键 Ctrl+Z
# sleep 120
```

```
#此时使用快捷键 Ctrl+Z
[1]+  Stopped                 sleep 120
#使用 jobs 命令查看后台任务
# jobs
[1]+  Stopped                 sleep 120
```

此时可以使用 fg 或 bg 命令，将任务放在前台或者后台继续运行。当然也可以向该进程发送信号 18 让其继续运行。

注意：随意终止系统中的进程可能会对系统的稳定性造成影响，因此在结束任务时应该特别注意。

8.2.8　查看进程优先级

对于单处理器系统而言，处理器同一时间只能计算一个任务。要让处理器同一时间处理多个任务，通常都采用分时处理的方式（这有点类似于通信模型中使用的时分复用）。系统将处理器的处理时间分为很多个小段（通常称为时间片，每个时间片非常短，对于人类而言基本可以忽略），在不同的时间片运行不同的任务，就可以让系统在同一时间处理多个任务。

在系统运行的所有任务中，有些任务需要获得更多的时间片，例如视频、音乐类进程、运算量巨大的进程等。为了能区分对待不同的任务，分时系统引入了优先级的概念，优先级高的进程可以获得更多的时间片。

大多数系统都使用一个数字表示优先级，Linux 系统中也是如此。Linux 系统中的优先级在-20 到 19 之间，一共有 40 个优先级，-20 表示最高优先级，19 表示最低优先级。

修改进程的优先级之前，需要查看进程的优先级。在 Linux 系统中，查看进程的优先级可以使用 ps 命令的选项 l：

```
#以长格式显示进程
# ps -l
F S   UID   PID  PPID   C PRI  NI ADDR SZ   WCHAN   TTY        TIME  CMD
4 S     0  18606 18604  0  75  0  -    1166  wait   pts/0     00:00:00 bash
4 R     0  19713 18606  0  77  0  -    1052  -     pts/0      00:00:00 ps
```

在上面的示例命令输出结果中，NI 字段就是进程的优先级。从大多数进程使用的优先级可以看出，系统默认的优先级为 0。

如果要查看系统中所有进程的优先级，可以使用选项 e 和 f：

```
#使用选项 e 和 f 查看系统中所有进程的优先级
# ps -efl
F S UID PID PPID C  PRI  NI ADDR SZ  WCHAN       STIME  TTY    TIME CMD
4 S root    1    0   0  75   0 - 516   -         Oct08   ?   00:00:02 init
1 S root    2    1   0 -40   - -   0   migrat  Oct08   ?   00:00:00 [mig]
1 S root    3    1   0  94  19 -   0   ksofti  Oct08   ?   00:00:00 [kso]
5 S root    4    1   0 -40  -   -   0   watchd  Oct08   ?   00:00:00 [wat]
……
```

8.2.9　指定进程运行优先级命令 nice

在 Linux 系统中，查看和管理进程的优先级通常使用命令 nice。nice 命令既可以查看系统默认的进程优先级，也可以设置进程运行的优先级别。本小节将简单介绍如何使用 nice 命令查看和设置程序运行的优先级。

【命令格式】

```
nice [option] [command]
```

如果不为 nice 命令指定任何选项和参数，命令将显示系统的默认优先级。

【常用选项】

nice 命令的常用选项只有一个 n，该选项用于指定程序或命令运行的优先级。

【用法示例】

（1）如果不为 nice 命令指定任何选项和参数，命令将显示系统默认优先级。例如使用 nice 命令查看系统默认优先级：

```
#使用 nice 命令查看进程默认优先级
# nice
0
```

（2）在讲解如何修改进程优先级时，此处引入一个计算型脚本文件 exam.sh，使用这个可执行的脚本文件讲解如何控制进程优先级。

运行脚本文件 exam.sh 并查看占用系统资源情况：

```
#运行可执行脚本 exam.sh，并将其放入后台执行
# ./exam.sh &
[1] 1198
#使用 top 命令查看进程占用系统资源情况
# top
top - 15:34:52 up 4 days, 18:53,  2 users,  load average: 1.84, 1.81, 1.56
Tasks:  83 total,   2 running,  81 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.8%us, 95.9%sy,  0.0%ni,  0.0%id,  0.0%wa, 0.0%hi, 0.3%si, 0.0%st
Mem:    515444k total,   406748k used,   108696k free,   108100k buffers
Swap:  1044184k total,        0k used,  1044184k free,   220852k cached

  PID USER    PR  NI  VIRT RES     SHR S %CPU  %MEM   TIME+   COMMAND
 1198 root    25   0  4484 1056    932 S 13.6   0.2  4:00.20  exam.sh
……
```

从上面的结果可以看出这个脚本文件占用的 CPU 资源相对其他进程而言，还是比较高的，可以通过改变优先级的方法调整其使用的系统资源。

（3）例如要以最低优先级运行脚本 exam.sh：

```
# nice -n 19 ./exam.sh &
[1] 28153
```

```
# ps -l 28153
F S   UID   PID   PPID  C PRI   NI ADDR SZ WCHAN TTY  TIME CMD
0 S    0  28153  8455  11 99   19  - 1121 pipe_w  pts/0 0:03  /bin/bash ./exam
```

从上面的结果可以看出，启动的进程优先级为 19。

此时系统中只有一个比较消耗资源的进程存在，无法对比，此时可以多添加几个进程。添加多个优先级不同的进程，然后使用 top 命令查看：

```
#使用 top 命令查看多个优先级不同的进程
# top
Tasks:  90 total,   8 running,  82 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.4%us, 95.6%sy,  1.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,
0.0%st
Mem:    515444k total,   409600k used,   105844k free,   109816k buffers
Swap:  1044184k total,        0k used,  1044184k free,   220856k cached

  PID USER PR   NI  VIRT RES  SHR   S %CPU %MEM   TIME+  COMMAND
15478 root   15 -10 4484 1052  932  S  8.6   0.2     0:03.87  exam.sh
12859 root   24 5   4484 1052  932  S  2.0   0.2     0:01.80  exam.sh
14693 root   24 0   4484 1052  932  R  1.7   0.2     0:00.96  exam.sh
13903 root   29 10  4484 1056  932  R  1.3   0.2     0:00.73  exam.sh
15858 root   15 0   2196 1000  800  R  0.3   0.2     0:00.11  top
28153 root   34 19  4484 1052  932  S  0.3   0.2     0:56.66  exam.sh
……
```

从上面 top 命令的界面中，可以看出不同优先级的进程在获取系统计算资源时的区别。

8.2.10　改变进程优先级命令 renice

许多时候需要重新调整一个已经运行的进程的优先级，这时可以使用命令 renice。本小节将简单介绍如何使用 renice 命令改变进程的优先级。

【命令格式】

```
renice -n PID
```

使用 renice 命令时，需要提供进程的 PID 号。

【常用选项】

renice 命令的常用选项只有 n，用于指定新优先级。

【用法示例】

使用 renice 命令修改进程的优先级时，需要指定进程的 PID。例如要重新设置 PID 为 15478 的进程优先级：

```
#使用-5 指定进程的新优先级
# renice -5 15478
15478: old priority -10, new priority -5
# ps -l 15478
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY   TIME CMD
4 R    0 15478  8455   9  80  -5 -    1121 -       pts/0  1:18  /bin/bash ./exam
```

从上面的结果可以看出，进程的优先级已经被重新设置。

在许多 Linux 系统中通常不用手动设置进程的优先级，除非管理一个计算资源丰富的大型系统或多用户系统时。

学习本节时，由于许多内容都涉及操作系统、进程等较为复杂的内容，因此读者可以按自身的学习目的，选择性地学习本节中的内容。

8.3　计划任务命令 crontab、at

许多时候管理员都希望系统能够自动执行任务。自动执行的任务可能是在某个指定的时间，将某个重要的目录备份，或者是自动执行一些管理任务等。这个特殊的功能在 Windows 系统中被称为计划任务，Linux 系统中也提供了这个功能，本节将简单介绍 Linux 系统中的计划任务。

8.3.1　为计划任务提供支持

开始为系统建立计划任务之前，需要为系统添加相关设置，以确保计划任务能够正确运行。计划任务需要的支持主要包括两个方面：正确运行系统服务、准确的系统时间。本小节将介绍如何为计划任务提供支持。

1．正确运行系统服务

为系统添加计划任务之前，必须要确保计划任务相关的服务已经正确启动。为计划任务提供支持的系统服务是 crond 和 atd，因此必须保证这两个服务能随系统自动启动。

设置 crond 服务随系统自动启动：

```
#使用 chkconfig 命令设置 crond 和 atd 服务随系统自动启动
#设置完成后使用选项 list 验证设置
# chkconfig --level 35 crond on
# chkconfig --list crond
# chkconfig atd on
# chkconfig --list atd
atd             0:off   1:off   2:on    3:on    4:on    5:on    6:off
crond           0:off   1:off   2:off   3:on    4:off   5:on    6:off
```

设置 crond 服务的启动状态后，可能还需要手动启动服务：

```
#使用 service 命令手动启动服务 crond 和 atd
# service crond start
Starting crond:                                            [  OK  ]
# service atd start
Starting atd:                                              [  OK  ]
```

2．校准系统时间

在 Linux 系统中时间的准确性非常重要，如果系统时间不准确，可能会带来一些问题，

例如系统日志记录不准确、计划任务不能准时运行等，因此应该在执行计划任务时保持系统时间准确。

在本书的第 3 章中，已经简单地介绍了 Linux 系统中的日期时间，但由于时间的推移，系统时间也会逐渐变得不准确。使用 date 命令逐一修改系统时间，是非常不现实的，此时推荐使用 NTP 通过网络同步时间。

通过网络同步时间的命令是 ntpdate。使用这个命令需要一个有效的时间服务器地址。时间服务器地址通常可以通过网络搜索得到，也可以查询 CERNET 网络中心的时间服务地址列表，或直接使用中国国家授时中心地址。

- CERNET 网络中心时间服务地址列表：http://www.time.edu.cn/mem.htm
- 中国国家授时中心地址：210.72.145.44

目前许多大学和机构都设置有自己的 NTP 地址，也可以使用这些地址作为时间服务器地址。

在本例中使用 NTP 服务器的地址为 192.168.111.15，这是一个本地 NTP 服务器地址。使用 ntpupdate 同步系统时钟：

```
#使用 ntpupdate 命令从服务器 192.168.111.15 上同步系统时间
# ntpdate 192.168.111.15
14 Oct 15:12:39 ntpdate[14647]: adjust time server 192.168.111.15 offset
0.000377 sec
#使用 hwclock 命令的选项 w 将系统时间同步到硬件时钟
# hwclock -w
```

也可以将以上命令写入计划任务中，实现自动同步时间功能。

如果你的网络中有许多服务器需要使用 NTP 服务同步时间，管理员通常需要考虑建立一个本地的 NTP 服务器，以方便同步本地时间。

注意：使用 NTP 同步本地时间需要用到互联网，关于网络方面的详细内容将在第 10 章中详细讲解。

8.3.2　cron 计划任务格式

cron 是 Linux 系统中的计划任务调度程序，其功能是允许用户自定义计划任务。计划任务可在指定时间一次或多次执行任务（例如每天凌晨 4 点执行备份、每过 15 分钟收集系统磁盘信息等）。本小节将简单介绍 cron 计划任务的格式。

【计划任务格式】

```
* * * * * command
```

上面的格式中，最前面由空格分隔的几个星号“*”表示任务执行的日期和时间，后面的 command 表示要执行的命令或脚本。

日期和时间使用空格作为字段的分隔符，这些字段（从左算起）的含义如下。

- 第 1 个字段表示分钟，取值范围为 1～59。
- 第 2 个字段表示小时，取值范围为 0～23（即 24 小时时间制）。
- 第 3 个字段表示日期，取值范围为 1～31。

- ❑ 第 4 个字段表示月份，取值范围为 1～12。
- ❑ 第 5 个字段表示星期，取值范围为 0～6（0 表示星期天）。

使用上面的格式表示时间时，可以使用几个比较简单的特殊字符，这些特殊字符及含义如下。

- ❑ 减号“-”：表示一个范围。例如“1-5”在月份字段上表示 1 月至 5 月。
- ❑ 星号“*”：表示每个单位时间。例如日期字段的星号表示每天，小时字段则表示每小时等。
- ❑ 逗号“,”：表示一个列表。例如在星期字段中的“0,1,3,5”表示在每周的星期天、星期一、星期三和星期五。
- ❑ 井号“#”：表示一个注释行。
- ❑ 斜杠“/”：表示一个分隔时间段。例如在分钟字段使用“*/30”表示每隔 30 分钟。

使用日期和时间时，每个字段都必须使用空格隔开。

【计划任务示例】

下面是一些日期和时间格式的应用举例：

```
* 4 1 * * /file/backup/backup_full.sh
```

每月 1 日凌晨 4 点运行/file/backup/full.sh 脚本文件。

```
0 4 * * * /file/backup/backup.sh
```

每天凌晨 4 点运行脚本文件/file/backup/backup.sh。

```
*/15 19-24 * * 6,0 /file/net_tj.sh
```

在每周星期六、星期日的 19 点至 24 点，每隔 15 分钟运行脚本文件/file/net_tj.sh。

8.3.3　添加计划任务命令 crontab

学习了计划任务格式之后，就可以按前面介绍的格式添加计划任务了。添加计划任务需要使用命令 crontab，本小节将简单介绍如何使用 crontab 命令添加计划任务。

【命令格式】

```
crontab [option]
```

【常用选项】

计划任务命令 crontab 通常不需要使用参数。常用的选项如下。

- ❑ u：指定要操作的系统用户。
- ❑ e：编辑计划任务。
- ❑ r：删除计划任务。
- ❑ l：查看已经存在的计划任务列表。

【用法示例】

（1）添加一个计划任务，每隔 20 分钟与时间服务器 192.168.111.15 同步一次时间，可以使用 crontab 命令的选项 e 建立计划任务：

```
#使用选项 e 编辑用户的计划任务
```

```
# crontab -e
```

此时将会弹出一个交互式页面，实际为 Vim 编辑器，按 i 键插入以下内容：

```
#在用户的计划任务结尾添加以下内容
#每 20 分钟与时间服务器同步一次时间，并写入硬件时钟
#Synchronize the time every 20 minutes.
*/20 * * * * /usr/sbin/ntpdate 192.168.111.15 &>/dev/null ; /sbin/hwclock
-w &>/dev/null
```

在上面的这条计划任务中，先使用/usr/sbin/ntpdate 192.168.111.15 同步时间，并将命令的输出和错误使用重定向输出到系统垃圾池/dev/null 中。然后执行命令/sbin/hwclock -w 将系统时钟写入硬件时钟，并将命令的输出和错误重定向到系统垃圾池。

添加完成后，按下 Esc 键，在末行模式下输入:wq 保存并退出即可生效。

添加计划任务时，建议为每一个计划任务添加详细的备注，以方便后期维护。

注意：上面的计划任务中使用了命令的全路径，这是因为在 cron 中不提供环境变量，因此所有命令和脚本都应该使用全路径。

（2）另一个实际应用的例子，一台 Linux 系统中存放有学生考试生成的数据。每天中午 11：30-11：50 之间和 17：30-17：50 之间，工作人员会将各考场的考试数据上传至服务器的某个目录中。由于数据不能及时向上级考试机构传递，为保证数据安全，管理员制定了一个备份脚本 backup_exam.sh（脚本的相关知识将在后续章节中介绍），将修改时间小于一天的所有文件备份。为此需要添加一个计划任务，于每天中午 12：10 和 18：10 执行这个备份脚本。

执行 crontab -e 打开 cron 交互界面，添加以下内容：

```
#在用户计划任务中添加以下内容
#每天 12:10 和 18:10 运行备份脚本
#Backup important test data.
10 12,18 * * * /root/backup_exam.sh &> /dev/null
```

保存退出后，系统就会在每天 12：10 和 18：10 运行备份文件。

成功写入一个计划任务后，如果发现已经存在的计划任务有错误，可以继续使用命令 crontab -e 修改，直到计划任务能被正确执行为止。

（3）如要查看已经存在的计划任务，可以使用 crontab 命令的选项 l：

```
#使用选项 l 查看用户的计划任务
# crontab -l
#Synchronize the time every 20 minutes.
*/20 * * * * /usr/sbin/ntpdate 192.168.100.105 &>/dev/null ; /sbin/hwclock
-w &>/dev/null
```

（4）如要删除所有的计划任务，可以使用选项 r：

```
#使用选项 r 删除用户的所有计划任务，并使用选项 l 验证
# crontab -r
# crontab -l
no crontab for root
```

从上面的命令结果中可以看到，选项 r 将用户的所有计划任务都删除了。

如果存在多个计划任务，通常不推荐使用选项 r 删除，此时可以使用命令 crontab -e 进入编辑计划任务状态，删除相关计划任务条目。

8.3.4　备份及恢复计划任务

如果有很多计划任务列表，为避免人工操作的失误（选项 e 与 r 在键盘上的位置很接近），应该备份计划任务。本小节将介绍如何备份计划任务。

备份计划任务需要使用命令 crontab -l >backup_filename，即将 crontab 命令输出的任务列表重定向到备份文件。

例如要将 root 用户的计划任务执行备份：

```
#使用重定向的方法备份计划任务
# crontab -l >root_cron
```

上面的命令将 root 用户的计划任务备份在了当前目录的文件 root_cron 中。

导入备份时，可以直接使用命令 crontab backup_filename。删除计划任务，然后导入备份文件：

```
#使用选项 r 删除所有计划任务
# crontab -r
#使用 crontab 命令导入计划任务，并使用选项 l 验证
# crontab root_cron
# crontab -l
#Synchronize the time every 20 minutes.
*/20 * * * * /usr/sbin/ntpdate 192.168.100.105 &>/dev/null ; /sbin/hwclock
-w &>/dev/null
```

导入计划任务的备份时，如果有计划任务存在，导入的备份将会覆盖已经存在的计划任务。因此如果要添加或修改计划任务，建议先写入备份文件，然后再导入。

如果系统中有很多计划任务，通常不建议使用 crontab -e 编辑计划任务，应该新建或编辑备份，以导入备份文件的方式添加，以免误操作，破坏系统中的计划任务。

8.3.5　用户计划任务

默认情况下，系统中所有用户都可以使用 cron 计划任务，但有些管理员可能不希望某些用户使用计划任务，这时管理员就需要管理用户的计划任务。本小节将介绍管理用户计划任务的相关内容。

（1）root 用户可以使用选项 u 管理普通用户的计划任务。例如查看用户 wlh 的计划任务：

```
#使用选项 u 和 l 查看用户 wlh 的计划任务
# crontab -u wlh -l
#test
* 18 * * * /home/wlh/test.sh
```

root 用户也可以使用选项 e 和 r，编辑和删除用户的计划任务。

（2）默认情况下，系统会将用户提交的计划任务放在目录/var/spool/cron/中，并以用户名命名计划任务文件。root 用户可以借此查看哪些用户使用了计划任务，也可以通过修改这些文件的方式修改用户的计划任务。

（3）如果需要禁用某些用户的计划任务功能，root 用户可以在文件/etc/cron.deny 中添加需要禁用的用户名。

如果禁用了某个用户的计划任务功能，用户执行计划任务时，将会得到如下提示：

```
#普通用户添加计划任务时，将提示禁止使用相关功能
$ crontab -e
You (wlh) are not allowed to use this program (crontab)
See crontab(1) for more information
```

8.3.6　系统计划任务

在 Linux 系统中，系统默认安装的许多软件也使用了计划任务。但这些计划任务并不是以用户的身份添加的，因此无法通过 crontab 命令查看。通常将这些不是以用户身份添加的计划任务称为系统计划任务，本小节将简单介绍系统计划任务。

在系统配置文件目录中，存在一些很特殊的 cron 目录，查看这些目录可以使用以下命令：

```
#使用 ls 命令查看系统中的计划任务目录
# ls -ld /etc/cron* | grep "^d"
drwx------  2 root root 4096   Jul 15  2008    /etc/cron.d
drwxr-xr-x 2 root root  4096   Sep  8  22:40   /etc/cron.daily
drwxr-xr-x 2 root root  4096   Jul 15  2006    /etc/cron.hourly
drwxr-xr-x 2 root root  4096   Sep  8  22:35   /etc/cron.monthly
drwxr-xr-x 2 root root  4096   Sep  8  22:36   /etc/cron.weekly
```

上面的命令输出中，显示了 4 个非常关键的目录：cron.daily、cron.hourly、cron.monthly 和 cron.weekly。系统会分别在每天、每小时、每月、每周，执行放在这 4 个目录中的脚本文件。

提示： 细心的读者可以发现，这 4 个目录中已经存在一些文件，这些文件都是系统默认安装的计划任务。

root 用户可以向系统计划任务目录添加脚本，从而实现脚本不受用户计划任务影响的功能，主要用于软件或重要的管理脚本。

8.3.7　使用 at 执行一次性计划任务

在 Linux 系统中，除了可以使用 cron 执行一次性计划任务外，还可以使用 at 命令。at 命令通常用于添加一次性任务，这些任务通常可能是一个日程安排，也可能是临时备份一个目录中的数据等。本小节将简单介绍如何使用 at 命令执行一次性计划任务。

【命令格式】

```
at [option] [time command]
```

【常用选项】

- f：指定一个文件作为任务脚本。
- m：完成任务之后给用户发送一个邮件。
- l：列出计划任务列表。
- d：删除指定的计划任务。

【参数说明】

at 命令能够使用的时间格式非常多，可以使用标准的日期时间格式，也可以使用一些模糊的词语。可以接受的时间格式如下。

- at 19:00：这种方式没有指定日期，默认为当天 19:00，如果写入任务时系统时间已经过了 19 点，系统则默认为第 2 天的 19 点。
- at 7pm Oct16：使用这种方式指定日期时间时，默认为当年的 10 月 16 日，如果写入任务当天已经过了 10 月 16 日，则自动清除任务。
- at now + 2 hour：从现在算起两小时之后。
- at 10:10am tomorrow：明天早上 10 点 10 分。

读者可以参考上述范例，使用更多时间格式。

【用法示例】

（1）例如要添加一个计划任务，任务时间是明天早上 10 点 10 分，任务内容是将普通文件/file/backup 复制到目录/file/nfs/：

```
#使用 at 命令添加一个计划任务
# at 10:10am tomorrow
#由于命令未添加执行的命令和脚本，at 命令使用提示符提示用户继续输入
at> cp -R /file/backup /file/nfs/ &>/dev/null
at> <EOT>
job 7 at 2010-10-16 10:10
```

在提示符后输入要执行任务的时间后按 Enter 键，提示符 at>将提示用户输入要执行的命令，这时可以输入需要执行的命令和脚本。

如果有多个命令，可以在输入完成后，按 Enter 键继续输入。所有命令输入结束后，可以在新行中使用快捷键 Ctrl+D 结束输入。按下 Ctrl+D 组合键后，at 命令将显示<EOT>退出，并提示新任务已经加入。

从上面输入的命令中可以看出，at 与 cron 还有一个很大的不同点，即 at 支持使用现有的环境变量，而 cron 则不支持。

（2）如果要执行的任务是一个脚本文件，需要使用选项 f 指定要执行的脚本文件。例如：

```
#添加一个 5 小时后执行脚本 exam.sh 的任务
# at now +5 hour -f /root/exam.sh
job 10 at 2010-10-16 00:24
```

从上面的提示可以看到，任务已经成功添加。

虽然 at 命令可以支持现有的环境变量，但要使用它执行一个脚本文件时，仍然应该使用脚本的绝对模式。上面这个 at 命令示例，也可以使用如下符号：

```
#使用 at 命令添加执行脚本时，使用家目录符号~
# at now +5 hour -f ~/exam.sh
```

（3）如果需要查看使用 at 命令添加的任务列表，可以使用选项 l：

```
#使用选项 l 查看 at 命令添加的任务列表
# at -l
5       2010-10-16 19:00 a root
9       2010-10-16 00:23 a root
11      2010-10-16 00:30 a root
6       2010-10-16 08:00 a root
8       2010-10-16 00:23 a root
7       2010-10-16 10:10 a root
10      2010-10-16 00:24 a root
```

命令按先后顺序输出了要执行的任务列表，列表中第 1 列为任务序号。

提示： 除了使用选项 l 查看任务列表外，也可以直接使用命令 atq。

（4）删除一个已经存在的任务时，可以使用命令 atrm。例如要删除任务 7：

```
#使用 atrm 命令删除序号为 7 的任务，并使用 atq 命令验证
# atrm 7
# atq
5       2010-10-16 19:00 a root
9       2010-10-16 00:23 a root
11      2010-10-16 00:30 a root
6       2010-10-16 08:00 a root
8       2010-10-16 00:23 a root
10      2010-10-16 00:24 a root
```

删除任务时，也可以使用 at 命令的选项 d。

（5）同 cron 一样，at 命令也将执行的任务和环境变量放入脚本文件，并将其放置在目录/var/spool/at/中。查看这些文件：

```
# ls /var/spool/at/
a0000501475bd4  a0000801475777  a0000a01475778  spool
a0000601475940  a0000901475777  a0000b0147577e
```

（6）如果要禁止某个用户使用 at 添加计划任务，可以在文件/etc/at.deny 中添加该用户的用户名。

当被禁用此功能的用户使用 at 命令时，将会出现以下提示：

```
#当未授权用户使用 at 命令时将出现未授权提示
$ at 9am
You do not have permission to use at.
```

注意： 无论使用 cron 添加计划任务，还是使用 at 添加计划任务，都应该对使用命令和脚本输出做妥善的处理。可以采取的方法有使用 mail 命令，以邮件的形式将命令输出邮寄给用户，也可以使用重定向的方法将输出重定向到一个文件。

8.4　日 志 管 理

一个正在运行的系统会执行许多操作，如安装软件、用户登录系统、向硬盘写入数据等，这些操作都会引起一些事件。这些事件有些是正常的，有些事件可能是系统出现了某些严重的问题，可能会对系统、数据和隐私等构成威胁。Linux 系统会将这些事件记录下来形成系统日志。

系统日志记录了所有对安全、稳定运行等可能构成威胁的许多数据。所以系统管理员都会通过查看系统日志的方法，了解系统在过去一段时间内发生的事情，以便于采取相应的措施。日志管理对每个管理员而言都十分重要。本节将简要地介绍 Linux 系统中的日志管理。

8.4.1　syslogd 守护进程及其配置文本

系统在运行中可能会产生许多事件，这些事件都会通过日志消息的形式交给日志守护进程。日志守护进程会对这些日志消息进行筛选、分类，然后存放到指定的日志文件中。日志守护进程就像一个安全生产记录员，将系统中产生的所有日志信息，按事先设定的规则一一记录在案，以便于发生问题时管理员能够看到这些记录。因此在学习日志管理之前，需要了解日志守护进程的工作方式。

RHEL5.3 使用的日志服务是 syslog（不同的发行版使用的日志服务可能会不同），这是一个标准的日志服务器程序，不仅可以记录本地系统产生的消息，还可以记录多个联网的服务器、交换机和路由器等设备的日志。

【日志服务配置文件】

为了了解 syslogd 日志服务的工作方式，可以查看其配置文件。查看 syslog 的配置文件内容如下：

```
#查看日志守护进程 syslogd 的配置文件内容
# cat /etc/syslog.conf
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.*                                                  /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none;cron.none                 /var/log/messages

# The authpriv file has restricted access.
authpriv.*                                               /var/log/secure

# Log all the mail messages in one place.
mail.*                                                   -/var/log/maillog
```

```
# Log cron stuff
cron.*                                                  /var/log/cron

# Everybody gets emergency messages
*.emerg                                                 *

# Save news errors of level crit and higher in a special file.
uucp,news.crit                                           /var/log/spooler

# Save boot messages also to boot.log
local7.*                                                /var/log/boot.log
```

上面是系统默认的日志服务 syslogd 的配置文件，以“#”开头的为注释行。注释行只为说明相关功能并不是一个有效的配置行，服务读取配置文件时，将忽略注释行。

配置文件中的有效行都有一个固定的格式，每一个有效行都以设备、级别（其格式为设备.级别）组成的选择域开始，以文件作为结尾。下面将讲解这些术语的作用。

【设备】

设备是用于描述消息来源的标识符，即用于筛选某一个应用程序或进程发送的日志消息。上面的配置文件中列出了一些常见的设备，例如 kern、mail、cron 等。常见的设备及其说明如表 8.2 所示。

表 8.2　常见的设备及其说明

设　　备	说　　明
kern	来自内核的消息
mail	来自邮件系统的消息
auth	与用户登录、验证密码有关的消息
authpriv	同 auth 一样，当记录消息时，将消息记录到只有相关用户能读取的文件内
cron	来自 cron 任务调度程序的消息
news	与网络新闻协议 NNTP 相关的消息
deamon	某些系统守护进程产生的消息
lpr	与打印机相关的消息
mark	定时发送消息的时标程序
ftp	与 FTP 守护进程有关的消息
ntp	与网络时间协议相关的消息
uucp	与 uucp 子系统相关的消息
local0-local7	7 种本地类型的消息

除了表 8.2 所示的设备之外，还可以使用其他一些不常见的设备（例如使用@表示的主机等），此处不再一一介绍，有兴趣的读者可以参考相关文档。

提示：如果使用的设备名称为星号“*”，则表示来自所有设备的消息。

【消息级别】

级别是消息的重要性标识，借助消息级别，日志守护进程 syslogd 可以判断消息是否重要。配置文件中可以使用的日志消息级别及其说明如表 8.3 所示。

表 8.3　日志消息级别及其说明

级　　别	功 能 说 明
debug	程序调试级别，除非调试程序，否则不要将日志记录设置为此级别，因为大量的消息将会很快充满磁盘
info	信息消息
notice	正常但值得引起注意的消息
warning	警告的消息
err	错误的消息
crit	严重的错误消息
alert	必须要立即采取措施的情况
emerg	紧急情况，系统可能会无法使用。这个消息将会发送给所有正在使用系统的用户
none	程序中未给出选择的伪级别

在上面的日志消息级别中，从上至下情况越严重，危险等级越高。在配置文件中使用消息级别时，只需要指定消息的最低级别即可。例如指定最低消息级别为 err，日志服务筛选时，将会筛选所有危险等级大于和等于此级别的消息。

提示：用户在设置记录日志消息等级时，通常不要使用 err 以下的级别，否则可能大量的消息很快就会充满整个磁盘，并且还会引起系统性能下降。

【选择域】

选择域由一个或多个设备和消息级别组成，用于筛选多个设备的消息。在默认的配置文件中，使用选择域的例子如下：

```
#使用选择域的行
*.info;mail.none;authpriv.none;cron.none            /var/log/messages
uucp,news.crit                                      /var/log/spooler
```

【动作】

除了设备、选择域和日志消息级别外，配置文件中还包含了动作。动作主要将匹配设备、日志消息级别及选择域的日志消息放入相应的日志文件中。动作位于有效行的行尾，通常是一个文件，例如系统日志文件/var/log/messages。

【工作原理】

从上面的讲解中可以看出日志守护进程的工作原理：日志服务 syslogd 从应用程序中发送过来的消息中，按配置文件定义的规则进行筛选，将符合条件的日志消息写入配置文件相应的日志文件中。

（1）下面是配置文件中比较典型的两行内容，以此作为示例讲解：

```
cron.*                                              /var/log/cron
*.info;mail.none;authpriv.none;cron.none           /var/log/messages
```

第 1 行将所有来自系统任务调度程序 cron 的消息放入日志文件/var/log/cron 中。第 2 行将所有消息中级别大于 info 的消息，来自邮件程序、用户验证、系统任务调度程序 cron 的未定义级别的消息都放入系统日志文件/var/log/messages 中。

（2）syslogd 日志服务是一个标准的日志服务器程序，也可以将日志消息发送给其他日志服务器。

例如将日志消息发送到其他日志服务器的配置项如下：

```
mail.*                                                  @192.168.1.10
```

在上面这个示例中，syslogd 会把所有收集到的有关邮件的日志消息发送给 IP 地址为 192.168.1.10 的服务器，由此服务器筛选并保存日志。

8.4.2　日志消息的格式

日志消息记录了系统或应用程序的运行情况，因此管理员有必要时常查看日志，以便获得系统及应用程序过去的运行情况信息。本小节将介绍日志消息的格式。

Linux 系统中几乎所有的日志消息都使用了一个固定的格式：

```
time hostname process version:information
```

这个固定格式中使用了两种字段分隔符，前面 4 个字段之间使用的分隔符是空格，最后两个字段之间使用的是冒号“:”。这几个字段及其含义如下。

- time：产生日志消息的时间。便于管理员判断消息产生时系统的状态和使用情况。
- hostname：产生日志消息的主机名。如果 syslog 记录的是多个主机的日志，这个字段便于判断产生日志的主机。
- process：产生日志的应用程序。如果记录的是应用程序产生的日志，还会记录应用程序的 PID。
- version：产生此日志的应用程序或系统服务的版本号。
- infomation：日志内容，根据不同的应用程序和产生的消息不同，这个字段描述了程序某些方面的内容，例如服务重启、系统关机、应用程序错误、身份验证失败、网络超时等。

查看当前系统中的日志：

```
#使用 tail 命令查看系统日志的最后几行
# tail /var/log/messages
Oct 17 04:02:29 localhost syslogd 1.4.1: restart.
Oct 17 13:45:42 localhost kernel: Removing netfilter NETLINK layer.
Oct 17 13:45:45 localhost kernel: ip tables: (C) 2000-2006 Netfilter Core
Team
Oct 17 13:45:45 localhost kernel: Netfilter messages via NETLINK v0.30.
Oct 17 13:45:45 localhost kernel: ip_conntrack version 2.4 (4096 buckets,
32768 max) - 228 bytes per conntrack
```

上面这几条日志中，第 1 条是日志服务重新启动的消息，后面几条是由内核发出的关于防火墙的消息。

8.4.3　记录日志消息命令 logger

Linux 系统提供了一个产生日志消息的命令 logger，该命令的功能是提交一条日志消息

给日志服务进程。日志服务进程会对日志消息进行筛选，然后写入日志文件。本小节将简单介绍如何使用 logger 命令产生日志消息。

一个典型的例子：编写与系统相关的脚本时，使用 logger 命令提交日志消息，让 syslogd 获取并保存到相应的日志文件中。这样做的目的是将脚本的日志写入系统日志中，以便于日志监视程序并将其及时反馈给管理员。

【命令格式】

```
logger [option] message
```

【常用选项】

- ❑ i：使用 logger 的 PID 作为消息的 PID。
- ❑ p：使用指定的选择域。
- ❑ t：使用指定进程的名称。

logger 命令还有一些其他选项，读者可以阅读说明文档。

【用法示例】

当编写一段较为重要的脚本时，可以使用 logger 命令将自定义的脚本提示消息写入日志消息中。

例如使用 logger 命令写入一条日志消息：

```
#使用 logger 命令添加一条日志消息
# logger -i -p local6.err -t test.sh "This is a test message."
#使用 tail 命令验证日志消息是否已写入
# tail -1 /var/log/messages
Oct 25 22:41:55 localhost test.sh[19790]: This is a test message.
```

从上面的示例命令的输出中可以看到，logger 命令向系统日志中写入了一条日志记录。

注意：日志服务器在运营维护过程中经常被用到，关于日志服务器的更多应用，感兴趣的读者可以参考相关文档。

8.4.4　日志轮循

随着系统运行的时间越来越长，syslogd 将新筛选的日志消息不断地写入指定的文件末尾。日志文件会变得越来越长，占用的磁盘空间也越来越多。另一方面，如果系统管理员要查看某一时刻的日志消息，翻阅一个长达数兆字节的日志文件是一件非常不现实的事。所以应该建立一个日志更新机制，将日志按时间进行整理，这样就可以解决日志文件过长的问题。这种日志更新机制就是日志轮循。

大多数 UNIX 和 Linux 系统中都安装有日志轮循工具，这些工具能够自动更新系统中的日志。在 RHEL5.3 中，使用的日志轮循工具是 logrotate。默认情况下，logrotate 以一个工作周为周期，备份旧的日志文件，删除过期的日志文件，并更换一个长度为零的日志文件，以便于日志守护进程重新记录日志。

在系统日志目录/var/log 中，有许多 logrotate 轮循日志时产生的文件。查看系统日志目

录中的日志文件如下：

```
#使用 ls 命令查看系统日志目录中的日志文件
# ls /var/log/
acpid               cron          maillog.2   rpmpkgs.2           spooler.2
anaconda.log            cron.1        maillog.3   rpmpkgs.3           spooler.3
anaconda.syslog         cron.2        maillog.4   rpmpkgs.4           spooler.4
……
```

在上面的日志文件中，“x.1”、“x.2”、“x.3”和“x.4”为 logrotate 按时间顺序产生的日志文件，其中 x 为日志文件的名称。

日志轮循工具 logrotate 是通过系统计划任务的方式调度的。可以使用以下命令查看其计划任务的文件：

```
#使用 ls 命令查看系统计划任务文件
# ls /etc/cron.daily/
0anacron     cups        makewhatis.cron     prelink     tmpwatch
0logwatch    logrotate   mlocate.cron        rpm
#使用 cat 命令查看 logrotate 计划任务文件
# cat /etc/cron.daily/logrotate
#!/bin/sh

/usr/sbin/logrotate /etc/logrotate.conf
……
```

从上面的命令输出中可以看出，日志轮循程序 logrotate 每天都会检查其配置文件中的配置，以决定是否要在当天进行日志轮循工作。

提示： 通常 logrotate 只会轮循已经配置过的日志文件。如果要轮循未配置的日志文件，可以修改其配置文件/etc/logrotate.conf，或在目录/etc/logrotate.d 中添加新日志文件的轮循方案。

8.4.5　监视系统日志

对于大多数系统管理员而言，业务系统持续运行必然会产生许多日志。由于日志守护进程在筛选日志时，并不知道哪些日志应该引起管理员的注意，只将规定级别的日志消息写入日志文件中，因此管理员发现一些重要的日志消息时，可能日志事件已经过了许久而变得毫无意义。

打开许多个终端，并在这些终端下执行 tail -f 命令，实时地查看日志文件的变化情况。阅读每一条由日志服务程序 syslogd 筛选的日志是非常不现实的，将消耗巨大的人力。

基于以上原因，管理员可能需要一个能够从海量的日志文件中筛选出重要日志并及时报告的“财务式的管家”。这个“管家”能够代替管理员做出一些决定，将重要的信息从日志文件中筛选出来，并将这些重要的日志信息汇总产生一个“报表”。及时地将这些“报表”交给管理员，管理员通过这些“报表”就可以获取系统和应用程序的详细信息。

这个智能的“管家”称为日志监视软件，利用日志监视软件，管理员可以轻松地监视系统日志。

在 Linux 系统中，有许多优秀的日志监视软件。其中最流行的日志监视软件主要有两个：Logwatch 和 Swatch。其中 Logwatch 是 RHEL5.3 中默认安装的日志监视软件，本小节将简单介绍 Logwatch 的用法。

默认情况下，RHEL5.3 中的日志监视软件 Logwatch 会统计每天产生的日志信息，并以邮件的形式将统计后的日志信息发送给 root 用户。root 用户只需要通过查看这些邮件，就可以了解系统一整天的基本情况。

下面是一封由 Logwatch 发给某个文件服务器 root 用户的邮件内容：

```
#由 Logwatch 产生的邮件
From root@FilSvr02  Mon Jul 12 04:03:07 2010
……
#省略部分为邮件头
#以下为 Logwatch 头信息，需要注意的部分是统计日志的日期
 ################### Logwatch 7.3 (03/24/06) ####################
       Processing Initiated: Mon Jul 12 04:02:07 2010
       Date Range Processed: yesterday
                             ( 2010-Jul-11 )
                             Period is day.
     Detail Level of Output: 0
          Type of Output: unformatted
         Logfiles for Host: filsvr02
  ##################################################################

#以下为 Logwatch 的分类日志，不同的系统以下部分可能会有所不同
#Logwatch 使用服务分类不同的日志
#此处显示的日志是 Logwatch 认为十分重要的日志
 --------------------- samba Begin ------------------------

 ---------------------- samba End -------------------------

 --------------------- sendmail Begin ------------------------

 ---------------------- sendmail End -------------------------

 --------------------- XNTPD Begin ------------------------

 Total synchronizations 4 (hosts: 2)

 ---------------------- XNTPD End -------------------------

#除服务产生的日志外，Logwatch 还会发送文件系统的使用情况
 --------------------- Disk Space Begin -------------------------

 Filesystem            Size    Used    Avail   Use%    Mounted on
 /dev/hdb1             17G     3.3G    13G     21%     /
```

```
 /dev/md0            302G    19G     269G    7%      /file
 /dev/hdb2           19G     173M    18G     1%      /file/hdb2

 ---------------------- Disk Space End -------------------------

 ###################### Logwatch End #########################
```

可以通过上面的信息了解在过去的一天中，系统中所有日志的基本概况。在上面这个邮件示例中可以看出系统一切正常。

许多时候管理员需要立即查看所有日志文件，这时可以使用 Logwatch 立即扫描日志文件产生监视信息：

```
#使用 print 选项立即扫描日志并产生监视信息
# logwatch --print
#该命令的输出与邮件内容十分相似
 ################### Logwatch 7.3 (03/24/06) ####################
       Processing Initiated: Sun Oct 17 21:27:25 2010
       Date Range Processed: yesterday
                            ( 2010-Oct-16 )
……
  ##################################################################

 --------------------- samba Begin ------------------------
……
 --------------------- Disk Space Begin ------------------------

 Filesystem          Size    Used    Avail   Use%    Mounted on
 /dev/hdb1           17G     4.2G    12G     27%     /
 /dev/md0            302G    178G    109G    63%     /file
 /dev/hdb2           19G     927M    17G     6%      /file/hdb2
……
```

使用 print 选项时，Logwatch 将会立即统计系统中的日志消息，并输出到用户所在的终端。由于使用不同的系统及服务，因此上面的命令在不同的服务器上使用时，得到的结果也会有所不同。

日志监视软件大大减少了系统管理员查看日志文件的工作。关于 Logwatch 及其他日志监视软件的应用，读者可以查阅相关文档了解，此处不再赘述。

注意：在本小节中仅简单介绍如何使用 Logwatch 监视系统日志，关于 Logwatch 的更多用法，感兴趣的读者可以阅读相关文档。

8.5　小　　结

- 8.1 节介绍了 Linux 系统中的服务管理。在 Linux 系统中，许多内部功能、业务系统都通过服务的方式实现，因此应该掌握如何管理这些服务。
- 8.2 节讲解了 Linux 系统中的进程管理，包括查看进程、进程树、控制进程等。对

于许多初学者而言，查看、杀死进程、进程优先级控制等都属于需要了解的内容。

- 8.3 节主要介绍了如何利用 cron 和 at 在指定时间执行计划任务，许多管理员都使用 cron 和 at 实现定时备份数据等自动化任务。计划任务是一个非常重要的内容，初学者应该掌握此部分内容。
- 8.4 节介绍了 Linux 系统中的日志管理，包括日志服务、配置文件、日志服务的工作原理、日志轮循工具和日志监视工具的使用方法。读者可以从这一节的讲解中了解 Linux 日志消息机制，以及如何使用系统自带的日志工具了解系统运行情况等内容。此节中的 logger 命令经常用于较大型的脚本编程，因此需要特别注意其用法。

本章中详细介绍了 Linux 系统的服务、进程、日志等内容，其中的许多命令都会在实际应用中用到，初学者应该对这些命令及用法多加练习。

第 9 章　数据备份和应用程序管理

在数据库服务器、文件服务器等关键性服务器中，数据的安全非常重要。如果这些系统出现问题，一次数据灾难（重要数据大面积丢失称为数据灾难）就有可能导致巨大的损失。为了避免重要数据的损失，大多数企业、机构和政府机关都为这些数据做了非常详细而有效的备份策略。

本章将简单介绍系统数据安全及备份策略相关的基础知识、Linux 系统中的数据备份、应用程序管理等，涉及的主要内容如下。

- 介绍数据备份的重要性、存储介质及备份类型等基础知识。
- 介绍 Linux 系统中最常用的数据备份工具 tar、gzip、cpio、dd 及压缩工具等。
- 使用 RPM 包管理器管理 Linux 系统中的应用程序。
- 安装编译环境、获取应用程序源代码和应用程序的编译安装等。
- RHEL5.3 中的 yum 应用程序管理器介绍，利用 yum 的本地源安装和管理应用程序。

9.1　数据备份基础

近年来，重要数据丢失带来巨大损失的例子举不胜举。例如 2001 年美国 9·11 事件以后，许多公司都倒闭了，倒闭的原因是这些公司失去了他们赖以生存的数据，例如保存在计算机中的客户资料、投资公司的账目等。

由于这些惨痛的教训，人们越来越重视数据备份。数据备份是指将保存在计算机中的重要数据，按科学的方法制作一个拷贝。当原始数据丢失或损坏后，可以利用这个数据的拷贝进行恢复。数据备份可以使数据丢失带来的损失降至最低。本节将简单介绍数据备份的基本方法。

9.1.1　数据备份概述

对于许多管理着重要数据的管理员而言，总担心这些重要数据会丢失。本小节将讨论导致数据丢失的原因、保护数据的方式等内容。

1．数据丢失的原因

导致数据丢失的原因有很多，下面列举了一些可能的原因。

- 人为误操作、黑客破坏等人为因素。例如错误地执行了 rm、mkfs 等命令。
- 供电中断，系统来不及将数据回写到磁盘中。
- 磁盘机械结构损坏、使用寿命等技术因素。
- 地震、火山爆发等无法控制的自然因素。

- 火灾、盗窃等损坏计算机硬件的行为。

基于以上原因，为自己管理的服务器数据制定一个非常详细而全面的数据备份策略是很有必要的。备份策略的内容包括备份的数据、备份的方式、执行备份的时间，以及备份数据的保存方式等。当数据丢失或损坏时，一个周详的数据备份计划就可以充分地显示它的价值了。

2．如何避免数据丢失

对于许多人而言，丢失数据都是一件非常痛苦的事，如果是家庭用户丢失了数据，可能会损失一些辛苦收集的资料，或者是在金钱上受到一定的损失。如果数据丢失发生在企业，轻则影响企业的业务，重则可能会导致企业倒闭。因此应该采取一定的措施避免数据丢失。

- 在系统架构上使用容错机制，从技术上尽量避免发生数据丢失。例如使用 RAID 5、RAID 10 等容错机制避免磁盘损坏带来的数据丢失。
- 使用 UPS 电源（一种电源保护设备，当市电停止后，UPS 将在瞬间启动替代市电，以免计算机瞬间停机导致数据丢失）、发电机等设备保持关键服务器不间断运行。
- 为机房添加制冷设备，监控机房温度、湿度、空气洁净度等，为计算机运算创造良好的环境。安装防静电地板，防止静电损坏计算机硬件。
- 为防止地震、火山爆发等自然因素破坏，通常建议异地保存备份。两个备份数据在地理位置上至少相距 100 公里以上。
- 建立一个合理的数据备份策略，将可能的损失降到最低。

除了以上措施外，还需要做好防火、防盗等措施。

9.1.2　备份数据存放的介质

存放介质的好坏直接影响了备份数据的安全，因此在制定数据备份策略时，必须要考虑使用何种介质保存数据备份。数据及业务类型和保存时间的长短决定了用于存放备份数据的介质类型。

下面将简单介绍一些常用于存放备份数据的介质。

- 磁带：使用磁带存放备份数据是很古老的方法，使用这种方法存放备份数据通常可以保存很久（20 年以上），但这种方式一次性投入很大。
- CD/DVD 光盘：CD、DVD 光盘是近年来兴起的比较常用的存放备份数据的介质。这种方式投入较小，存放的数据保存时间也较长（按光盘的质量时间在 5～20 年之间）。
- 网络、硬盘存储：由于网络和硬盘存储使用起来非常方便，现在越来越多的企业也倾向于使用这种方式存放备份数据。

在制定灾难备份时，应该按照数据类型选定存放备份数据的介质类型。通常推荐使用保存期较长的介质存放不会更改的数据，例如需要输入计算机的档案、材料等。使用保存期较短的介质保存经常会变动的备份数据，例如一些客户资料、数据库资料等。

9.1.3　备份类型

制定备份策略时，应该搭配使用几种备份类型，这是出于几个方面的考虑。首先需要

考虑的是用于存放备份数据的介质，如果备份的数据量非常大，存放备份数据时将会消耗大量的存储空间；其次需要考虑的是对数据备份、恢复时花费的时间等。

一个合理的数据备份策略，应该详细地制定多个备份类型使用的周期，在一个周期内使用多种备份类型相配合以达到节约存储空间、加快备份、恢复数据时速度等目的。

下面将介绍常用的备份类型及其特点。

- 完全备份：备份数据时，将指定位置的所有数据备份，不管这些数据上次是否已经备份。完全备份实际上就是将目标位置的所有文件打包成一个文件。
- 差异备份：差异备份是将指定位置的自上次备份以来变化过的数据进行备份。由于差异备份只备份变化过的数据，因此差异备份产生的数据量通常比完全备份要小很多。
- 增量备份：增量备份时，将指定位置的自上次备份以来变化过的数据进行备份，而不考虑上次备份的类型。

在上面的几种备份类型中，最容易让人混淆的概念是差异备份和增量备份，要讲清楚这两个概念，就必须要了解备份软件的工作原理。备份软件在使用差异备份和增量备份时，通常会检查文件的更新标记。

更新标记通常是文件系统给文件附加上的特殊标记，在文件被修改之后，文件系统通常会自动给文件附加更新标记。这个更新标记在 Windows 操作系统中通常是“存档文件”标记（NTFS 文件系统的文件高级属性中，可以查看到存档文件标记），而在 Linux 系统中通常使用时间戳记标志文件的修改时间。

差异备份和增量备份的最大区别是，差异备份不会清除文件的更新标记，而增量备份会清除更新标记。从此可以看出，增量备份是一个增量的过程（几个连续的差异备份中，最后一个差异备份是前几次所有差异备份之和）。而增量备份的每次备份都是上一次增量备份的补充。

使用完全备份和增量备份时，备份软件都会清除已备份文件的更新标记。在 Windows 系统中，备份软件会将已备份的文件的存档标记清除，而 Linux 系统中则不会清除文件的时间戳记。

说明：本例中以 Windows 和 Linux 自带的备份软件 NTbackup 和 tar 为例，说明备份类型的差异。

3 个备份类型的功能及优缺点如表 9.1 所示。

表 9.1　3 种备份类型间的比较

备份类型	功　　能	优　　点	缺　　点	清除更新标记
完全备份	备份所有目标文件	恢复数据时比较方便	备份需要的时间长，占用备份介质最大	清除
差异备份	备份自上次备份以来更新的文件	恢复数据速度比增量备份快	占用空间比增量备份多，备份速度比增量备份速度慢	不清除
增量备份	备份自上次备份以来更新的文件	备份速度最快	恢复数据时间比差异备份和完全备份慢	清除

制定备份策略时，通常使用 3 种备份相结合的方式，例如在周一对数据做一次完全备份，周二至周五每天做一次差异备份，周六周日每天做一次增量备份等。

注意：不同的备份软件，使用标记文件更新的方法和标记都有所不同，因此如果使用专用的备份软件，应该阅读相关文档。

9.1.4　备份时间选择

在制定数据备份策略时，还要考虑数据备份对在线业务带来的影响。数据备份会产生大量的磁盘读写操作，因此会对系统造成不小的压力。如果备份对象是数据库，过多的数据记录写入，还可能会造成备份数据不一致；如果备份对象是普通的业务数据，过多的磁盘数据写入也不利于统计文件变化。

基于以上因素，通常建议在业务压力较小的时间执行备份。如果系统业务属于对全国的公众服务，通常选择在凌晨 3 点至 5 点等使用业务较少的时间段内执行备份操作。如果系统业务仅仅是对公司内部服务，并且数据量较小，可以在员工午休或就餐时间执行备份。

进行数据备份时，一般都使用 cron 计划任务等自动工具，对预先设定的目录或文件进行备份，这样既可以减少人为工作量，又可以选择在系统压力较小的时间段进行操作。

一个比较周详的备份计划还包括数据备份之后的验证及保存备份数据的措施等。通常建议在数据备份后，每隔一段时间对备份的数据执行一次验证，以确保备份数据准确无误。

9.2　tar 备份工具

目前许多 Linux 系统都安装在服务器上，大多数服务器都会保存有非常重要的数据，因此几乎所有的 Linux 发行版都安装了基本的数据备份工具。虽然这些工具只是一些最基本的工具，但其功能仍然十分强大，足以应付多数数据备份任务。

系统管理员可以利用这些系统自带的工具对重要的数据进行备份，当然也可以使用商业数据备份软件。在系统自带的备份工具中，最常使用的工具是归档工具 tar。本节将简单介绍 Linux 系统自带的数据归档工具 tar。

9.2.1　tar 命令的基本格式

在 UNIX、Linux 系统中，有许多命令可以用于备份数据，其中最常用的命令是 tar 命令。tar 是 UNIX 和 Linux 系统中的打包工具，可以将多个文件或目录打包（也称为归档）成一个文件，因最初设计目的是用于磁带备份（tape archive）而得名。

现在 tar 打包的文件格式已经成为一种标准。在网络上下载的软件安装包、文档等通常都使用了 tar 打包，因此初学者非常有必要学习这个工具的使用方法。

【命令格式】

```
tar [option] [file]…
```

【常用选项】

tar 命令的选项十分多，但其常用的选项却很少。常用选项及其含义如下。

- c：建立一个新归档文件。
- f：指定需要归档的文件。
- t：列出指定文件的内容。
- v：以详细模式显示命令执行过程。
- x：从归档文件中还原文件。
- z：通过 gzip 处理归档的文件。
- j：通过 bzip2 处理归档的文件。

tar 还有许多选项，此处不再一一介绍，读者可以阅读其帮助文档。

提示：使用 tar 时，一定要理解归档文件与 Windows 系统中的压缩文件的区别，不要混淆二者。压缩文件将在 9.4 节中介绍。

9.2.2　tar 归档和备份文件

tar 命令的功能是将多个文件及目录归档为一个文件，因此也可以使用这个命令对文件和目录进行备份。本小节将介绍如何使用 tar 命令归档文件及目录。

使用 tar 命令对文件进行归档时，需要使用选项 c 和 f 创建新的归档文件，并指定要归档的目录。

【归档文件格式】

```
tar -cf filename directory
```

上面的命令中，filename 是归档之后生成的文件名称，directory 为要归档的目录或文件。

【用法示例】

（1）例如要归档 root 用户的家目录：

```
#使用选项 c 和 f 归档 root 目录，并将归档文件命名为 root.tar
#假定当前工作目录为根目录/
# tar -cf root.tar root
```

在根目录中使用此命令就可以备份 root 用户的家目录。

（2）执行命令时，tar 会占用终端，此时可以在命令后加上&，将命令放到后台执行：

```
#使用&将命令放在后台执行
# tar -cf root.tar root &
```

（3）也可以使用选项 v，显示归档时的详细信息：

```
#使用选项 v 显示归档详细信息
# tar -vcf root.tar root
```

（4）在归档文件时，有时可能希望在归档文件的同时压缩归档的文件，这样就可以节省存储空间。

归档并压缩可以配合选项 z，使用 gzip 压缩归档的文件：

```
#使用选项 z，用 gzip 压缩归档的文件
# tar -zvcf root.tar.gz root
```

（5）如果在归档文件时，需要使用 bzip2 压缩归档文件，可以配合使用选项 j。例如归档 root 的家目录，并使用 bzip2 压缩归档文件：

```
#使用选项 j，用 bzip2 压缩归档的文件
# tar -jvcf root.tar.bz2 root
```

gzip 和 bzip2 都是 Linux 中最常用的压缩软件，将在本章的 9.4 节中介绍。

注意：使用 tar 命令创建归档文件时，通常使用以.tar 结尾命名生成的归档文件。以.tar.gz 结尾表示使用 gzip 压缩的归档文件，以.tar.bz2 结尾表示使用 bzip2 压缩的归档文件，以便于使用者恢复归档文件。

9.2.3　查看归档文件中的文件列表

许多时候希望查看归档文件中的文件列表，可能是查看文件中是否包含某个特定的文件，也可能是查找某个文件是否被归档。本小节将介绍如何查看归档文件中的文件列表。

要查看归档文件中的文件列表，可以配合使用选项 t。

（1）例如查看归档文件 root.tar 内的文件列表：

```
#使用选项 t 查看归档文件内的文件列表
# tar -tf root.tar
root/
root/.eggcups/
root/.ICEauthority
……
```

（2）如果要查看的归档文件使用了 gzip 压缩，应该同时配合使用选项 z。查看归档文件 root.tar.gz 内的文件列表：

```
#使用选项 z 和 t 查看使用 gzip 压缩的归档文件
# tar -tzf root.tar.gz
```

（3）查看使用 bzip2 压缩的归档文件时，应该同时配合使用选项 j。例如要查看归档文件 root.tar.bz 内的文件列表：

```
#使用选项 j 和 t 查看使用 bzip2 压缩的归档文件
# tar -tjf root.tar.bz
```

目前大多数版本的 tar 命令都支持不使用参数 z 和 j，查看使用 gzip 及 bzip2 压缩的归档文件，但通常都建议使用这两个参数。

9.2.4　从归档文件中还原文件

归档和备份文件的目的都是为了能够在需要时还原文件。有时从互联网上下载的文件

都是归档文件，这时也要从这些归档文件中将文件还原出来。本小节将简单介绍如何从归档文件中还原文件。

从归档文件中还原文件需要使用 tar 命令的 x 选项。

（1）将归档文件 root.tar 中的文件还原到当前目录下：

```
#使用选项 x 还原归档文件
# tar -xf root.tar
```

（2）如果需要在还原归档文件时，查看命令执行过程，可以加上选项 v：

```
#使用选项 v 显示命令执行过程
# tar -vxf root.tar
root/
root/.eggcups/
root/.ICEauthority
……
```

（3）还原使用 gzip 压缩的归档文件时，需要配合使用选项 z。例如将归档文件 root.tar.gz 还原到当前目录：

```
#使用选项 z 还原使用 gzip 压缩的归档文件
# tar -zxvf root.tar.gz
root/
root/.eggcups/
……
```

（4）如果要还原使用 bzip2 压缩的归档文件，则需要配合使用选项 j。例如将归档文件 root.tar.bz2 还原到当前目录：

```
#使用选项 j 还原使用 bzip2 压缩的归档文件
# tar -jxvf root.tar.bz2
root/
……
```

使用 tar 命令还原文件在 Linux 系统中经常使用，初学者应该熟悉其用法。

使用 tar 命令从压缩的归档文件中还原文件时，也可以不使用选项 z 和 j 还原压缩的归档文件，但通常建议使用这两个选项。

9.3　cpio 备份命令

cpio 是一个比较古老的备份命令，也是用于磁带机备份的工具。虽然如此，现在许多时候仍然需要使用这个命令，例如定制系统内存映像盘时等。本节将简单介绍 cpio 命令的使用方法。

小知识： 系统内存映像盘通常位于引导分区/boot 中，文件名以 initrd 开头。该文件主要用于系统启动时加载需要的模块，例如文件系统模块，RAID 模块等。

9.3.1　cpio 命令的基本格式

cpio 命令与 tar 命令一样，其功能也是将文件或文件列表归档到文件中。本小节将简单介绍 cpio 命令的基本格式。

【命令格式】

```
cpio [option] [destination-directory]
```

【常用选项】

- i：使用 copy-in 模式，还原归档文件或列归档文件中的文件列表。
- o：使用 copy-out 模式，建立归档文件。
- p：使用 copy-pass 模式，将文件直接复制到目的目录。
- c：使用老式的 ASCII 归档格式。如果需要跨平台使用，应该使用老式的 ASCII 归档格式。
- d：创建需要的目录。如果需要文件不处于同一目录中，应该使用此选项。
- v：显示处理过程的详细信息。
- t：显示归档文件中的文件列表。
- m：保持文件的时间戳记。
- H：使用指定的格式归档文件。

cpio 命令还有许多选项，此处不再赘述，读者可以通过阅读帮助文档了解和使用这些选项。

【参数说明】

在讲解 cpio 命令的选项时，讲到了 cpio 的 3 个模式，含义如下。

- copy-in：从归档文件包中恢复文件，或列出归档文件包的内容。使用此模式时，任何不是选项的参数都会被当作通配符，只有被匹配的文件才能复制出来。如果没有任何通配符，cpio 将恢复所有文件。
- copy-out：此模式会从标准输入读取文件列表，并将文件添加到归档文件中，最后将归档文件输出到标准输出。由于文件列表使用换行符作为分隔符，因此文件列表最好使用 find 命令生成。
- copy-pass：此模式会将文件列表中的文件复制到另一个目录，中间不使用归档包。copy-pass 模式相当于 copy-in 和 copy-out 模式的结合。

在 3 个模式中，copy-pass 模式较少使用，因此本书中仅介绍前两种模式，感兴趣的读者可以自行阅读相关文档，了解其用法。

cpio 命令可以支持许多归档文件格式，例如 binary、old ASCII、new ASCII 等。本书仅讨论其在 Linux 系统中的应用。跨平台使用时，读者应该阅读额外的文档，以了解其用法。

9.3.2　使用 cpio 归档文件

使用 cpio 命令归档文件时，需要使用 copy-out 模式（即选项 o），读取文件列表并归

档文件。除此之外，需要使用 find 命令为其生成需要归档的文件列表。

（1）由于 cpio 命令不会操作文件，因此输入、输出都必须要借助重定向或管道来完成。例如将当前目录中的所有文件归档：

```
#使用 find 命令输出当前目录中的所有文件
#使用选项 o 备份管道传过来的所有文件，并保存到上级目录中
# find -print | cpio -o >../backup.cpio
50611 blocks
```

上面的示例命令中，如果不使用重定向输出，cpio 命令会将结果直接输出到标准输出。

（2）直接使用 cpio 命令归档文件过程中，不会有任何提示信息。如果需要显示归档的文件，可以使用选项 v：

```
#使用选项 v 显示归档文件的详细信息
# find -print | cpio -ov >../backup.cpio
.
./libpng-1.5.4
./libpng-1.5.4/libpngpf.3
./libpng-1.5.4/libpng15.la
……
```

命令将显示所有归档的文件名称，并在最后显示统计信息。

（3）如果重定向的目标是一个设备，cpio 会将归档文件存放到设备上。例如将归档文件存放到磁带机设备：

```
#使用重定向将归档文件存放到磁带机设备
# find -print | cpio -ov >/dev/st0
```

（4）与 tar 归档命令不同，cpio 并没有提供压缩功能。如果要压缩生成的归档文件，可以通过管道和相应的压缩命令。

例如使用 gzip 压缩生成的归档文件：

```
#使用 gzip 命令压缩 cpio 归档文件
# find -print | cpio -ov | gzip >../backup.cpio.gz
```

关于 gzip 命令的使用方法，将在下一节中介绍。

（5）也可以使用 bzip2 压缩生成的归档文件：

```
#使用 bzip 命令压缩 cpio 归档文件
# find -print | cpio -ov | bzip2 >../backup.cpio.bz2
```

关于 bzip2 命令的使用方法，将在下一节中介绍。

提示： 为了区别使用 cpio 命令生成的归档文件，文件名通常使用.cpio 作为结尾。如果是使用了 gzip 压缩的 cpio 归档文件，应该使用.cpio.gz 作为结尾，使用 bzip2 压缩则应该使用.cpio.bz2。

9.3.3　查看归档文件中的文件列表

有时需要查看归档文件中的文件列表，以确定需要备份的文件是否包含在归档文件

中。此时可以使用选项 t，列出归档文件中的文件列表。本小节将简单介绍如何查看归档文件中的文件列表。

（1）如果要查看归档文件中的文件列表，可以使用选项 t：

```
#查看归档文件中的文件列表
# cpio -t <backup.cpio
.
libpng-1.5.4
libpng-1.5.4/libpngpf.3
libpng-1.5.4/libpng15.la
……
```

（2）查看归档文件中的文件列表时，也可以使用选项 v，查看更详细的文件信息：

```
#使用选项 v，像 ls 命令的长格式那样查看文件信息
# cpio -tv <backup.cpio
drwxr-xr-x   3 root     root      0    Aug  9 18:48 .
drwxr-xr-x   4 root     root      0    Aug  9 18:47 libpng-1.5.4
-rw-r--r--   1 1004    ecryptfs   797  Jul  7 19:24 libpng-1.5.4/libpngpf.3
……
```

（3）查看归档文件中的文件列表时，如果命令中使用了选项以外的字符，cpio 会将其当作文件通配符。例如使用“*.c”查看归档文件中所有以.c 结尾的文件：

```
#使用通配符查看特定的文件
# cpio -tv "*.c" <backup.cpio
-rw-r--r--   1 1004     ecryptfs   75067 Jul  7 19:24 libpng-1.5.4/png.c
-rw-r--r--   1 1004     ecryptfs  165457 Jul  7 19:24 libpng-1.5.4/pngrtran.c
-rw-r--r--   1 1004     ecryptfs   19293 Jul  7 19:24 libpng-1.5.4/pngerror.c
……
```

9.3.4　恢复 cpio 归档文件

恢复使用 cpio 命令归档的文件时，需要使用 cpio 命令的 copy-in 模式（即使用选项 i）。与查看归档文件中的文件列表相同，此模式也会将所有选项以外的字符当成通配符。

（1）如果归档文件使用了不同的格式，cpio 会自动判断并恢复文件，无须再指定归档文件的格式。

例如使用选项 i 将文件从归档文件中恢复出来：

```
#使用选项 i，将文件恢复到当前目录
# cpio -i <../backup.cpio
11365 blocks
```

（2）使用选项 i 时，cpio 只会显示统计信息。如果要显示恢复文件的详细信息，可以使用选项 v：

```
#使用选项 v，显示恢复文件的详情
# cpio -iv <../backup.cpio
.
libpng-1.5.4
```

```
libpng-1.5.4/libpngpf.3
libpng-1.5.4/libpng15.la
……
```

（3）当使用 find 命令查找并归档文件时，有时可能会归档某个目录中的单个文件（例如按修改时间戳记查找文件时）。恢复这种归档文件会存在困难，因为 cpio 命令不会创建不存在的目录。此时可以使用选项 d，创建需要的目录：

```
#使用选项 d 创建需要的目录
# cpio -idv <../backup_file.cpio
```

（4）使用 cpio 命令恢复文件时，有时可能希望保持文件的原有信息。这时可以使用选项 m 保持文件的时间戳记：

```
#使用选项 m 保持文件的时间戳记
# cpio -imdv <../backup.cpio
```

（5）有时可能希望只恢复归档文件中的一部分文件，此时可以像查看文件那样使用通配符。例如恢复归档文件中所有以.c 结尾的文件：

```
#恢复文件时使用通配符
# cpio -idv "*.c" <../backup.cpio
libpng-1.5.4/png.c
libpng-1.5.4/pngrtran.c
libpng-1.5.4/pngerror.c
……
```

（6）cpio 命令的另一种用途是恢复内存映像盘，以便于修改其中的内容：

```
#使用 cpio 命令恢复内存映像盘
#命令先使用 zcat 解压，然后在使用 cpio 命令恢复归档文件
# zcat /boot/initrd-2.6.18-194.el5.img | cpio -imd
```

cpio 命令还有许多用法，此处不一一列举，读者可以阅读其他文档了解其用法。

9.4　压缩工具和整盘备份工具 dd

如果备份文件十分大，通常建议将其进行压缩，这样可以减小用于存放备份数据的存储空间，节约用于存储空间方面的成本。压缩文件可以使用 Linux 操作系统中默认提供的几个用于压缩文件的工具。本节将简单介绍 Linux 操作系统中的压缩工具和整盘备份工具 dd 的使用。

9.4.1　使用 gzip 压缩文件

在 Linux 系统中，压缩文件可以使用命令 gzip，对文件、其他命令的输出执行压缩。gzip 是 GNU zip 的缩写，这是一个自由软件，于 1992 年首次发行。本小节将简单介绍如何使用 gzip 命令压缩文件。

压缩文件后，其占用磁盘的空间将会减小，但这也会造成文件无法使用。例如文本文件压缩之后，文件将不能使用 cat 命令查看，也不能使用 Vi 编辑其内容。虽然如此，在对文件进行网络传输、临时存储、转存时，使用压缩会省去许多麻烦（例如减小传输的文件尺寸可以减少消耗的时间等）。

提示：压缩文件虽然可以减小其占用的磁盘空间，但如果压缩的是视频、图片、声音等文件，其效果却非常不明显。

【命令格式】

```
gzip [option] filename
```

【常用选项】

- c：压缩后将结果输出到标准输出（用户所使用的终端），并保留原始文件。
- d：解压缩文件。
- l：显示压缩文件的详细信息。
- r：递归地处理目录下的所有文件及子目录，或递归地解压缩文件内的目录和子目录。
- v：显示命令的执行过程。
- t：测试压缩文件。
- *num*：*num* 为数字 1～9，代表压缩时使用的压缩率。1 表示压缩速度最快、压缩率最小，9 代表压缩速度最慢、压缩率最大。如果不使用此选项，则默认使用压缩率 6。

【用法示例】

（1）使用 gzip 命令压缩时，可以不使用任何选项。例如要压缩当前目录中的文件 test.tar：

```
#直接使用 gzip 命令压缩文件 test.tar
# gzip test.tar
```

gzip 命令将 test.tar 文件压缩后输出为一个新文件，压缩后的新文件名为原文件名结尾加上.gz。

需要注意的是使用以上方式压缩完成之后，命令将会自动删除原文件。

（2）如果需要保留原文件，应该使用选项 c，还要使用重定向的方式保存压缩后的文件：

```
#使用选项 c 保留原文件，9 表示压缩率
#由于选项 c 会将压缩结果输出到标准输出，因此需要使用重定向方式保存压缩结果
# gzip -c9 test.tar >test.tar.gz
```

注意：由于压缩率越大，计算量越大，因此如果原文件过大，上面这个示例命令会导致命令执行时间过长。

（3）如果要处理的是一个目录及目录中的所有文件，可以配合使用选项 r 递归地压缩目录中的所有文件。

例如要压缩目录 test 及目录中的所有文件：

```
#使用选项 r 递归地压缩目录中的所有文件
# gzip -r test
```

使用上面的命令时，gzip 会将目录 test 中的所有文件、目录及子目录中的文件执行压缩并删除原文件。

使用选项 r 压缩目录时，也可以与选项 l 配合，显示其中的所有文件及其压缩文件的详细信息等内容。

（4）有时需要测试一个压缩文件是否正确，可以配合使用选项 t。例如要测试压缩文件 test.tar.gz：

```
#使用选项 t 测试压缩文件是否正确
# gzip -t test.tar.gz
```

如果压缩文件中没有任何错误，命令将不会返回任何结果。

（5）如果要显示压缩文件的详细信息，可以使用选项 l：

```
#使用选项 l 查看压缩文件的详细信息
# gzip -l test.tar.gz
         compressed        uncompressed  ratio  uncompressed_name
           20313650           171127345  88.1% test.tar
```

在上面的示例命令中，gzip 命令输出了压缩后的大小、未压缩时的大小、压缩比例及未压缩文件的名称。

（6）解压缩使用 gzip 压缩的文件时，需要配合使用选项 d。例如将当前目录中的文件 test.tar.gz 解压缩：

```
#使用选项 d 解压缩文件
# gzip -dv test.tar.gz
test.tar.gz:      1.4% -- replaced with test.tar
```

与 Windows 系统中的压缩软件不同，使用 gzip 压缩时，无法对多个文件执行归档。因此如果要压缩的是多个文件或目录，应该先使用归档命令将文件归档。

9.4.2　使用 bzip2 压缩文件

在 Linux 系统中，除了可以使用 gzip 命令对文件进行压缩外，还可以使用另一个开源工具 bzip2 压缩文件。与 gzip 等大多数压缩工具相比，bzip2 的压缩效率更高，但速度相对较慢。本小节将简单介绍如何使用命令 bzip2 压缩文件。

【命令格式】

```
bzip2 [option] filename
```

【常用选项】

- ❑ k：压缩、解压缩完成之后，保留原始文件。
- ❑ d：执行解压缩任务。
- ❑ t：测试压缩文件的完整性。
- ❑ *num*：*num* 为数字 1～9，表示压缩率级别。其中 1 表示压缩率最低、速度最快，9 表示压缩率最高、速度最慢。

- v：执行时显示执行的详细信息。

【用法示例】

bzip2 命令的使用方法与 gzip 大致相同，许多选项的功能也相似，下面将简单介绍这些选项的使用方法。

（1）可以不使用任何选项压缩文件：

```
#使用 bzip2 命令压缩文件 test.tar
# bzip2 test.tar
```

命令将在压缩完成之后删除原始文件，生成的压缩文件名称为原始文件名结尾加上.bz2。

（2）如果要让 bzip2 完成压缩任务后不删除原始文件，可以配合使用选项 k。例如使用 bzip2 压缩文件 test.tar，并保留原始文件：

```
#使用选项 k 压缩并保留原始文件
# bzip2 -k test.tar
```

（3）如果要指定压缩率，可以直接使用数字作为选项。例如以最高压缩率压缩文件 test.tar 以节省存储空间：

```
#使用选项 k9 以最高压缩率压缩文件，并保留原始文件
# bzip2 -k9 test.tar
```

（4）如果压缩的文件中保存有重要数据，完成后应该对压缩文件执行测试。要测试压缩文件的完整性，可以配合使用选项 t。

例如测试压缩文件 test.tar.bz2 的完整性：

```
#使用选项 t 测试文件 test.tar.bz2 的完整性
# bzip2 -t test.tar.bz2
```

如果文件通过完整性测试，将不会返回任何结果。

（5）解压缩文件时，可以使用选项 d。例如要解压缩文件 test.tar.bz2 并显示详细信息：

```
#使用选项 d 和 v 解压缩文件并显示详细信息
# bzip2 -dvk test.tar.bz2
  test.tar.bz2: done
```

9.4.3　整盘备份命令 dd

dd 是一个非常特殊的命令，其作用是从标准输入或文件中读取数据，并按指定的格式转换数据，然后输出。dd 命令非常像 Windows 系统中的分区、磁盘备份工具 Ghost。使用这个命令可以备份整个分区、磁盘，并且备份时可以不必关闭系统、卸载文件系统。本小节将简单介绍如何使用 dd 命令备份数据。

【命令格式】

```
dd <OPTION>
```

dd 命令不需要使用参数，所有参数都通过选项指定。

【常用选项】

- if：指定要读取的文件，默认为标准输入。
- of：指定要输出的文件，默认为标准输出。
- ibs：指定读取数据时的块大小，默认为 512 字节。
- obs：指定输出数据时的块大小，默认为 512 字节。
- bs：将读取、输出时的块大小一起指定。
- count：指定读取的区块数。

【用法示例】

管理员在许多时候都要使用到这个命令，其目的可能是需要备份一个分区、硬盘和软盘等存储设备中的内容，也可能是需要验证某个设置，需要产生一个指定大小的文件等。下面将以一些示例讲解如何应用 dd 命令。

（1）有时候管理员需要一个类似于 Ghost 的工具，备份一个分区甚至是一个磁盘，这时可以使用 dd 命令来完成。

例如要备份磁盘 sda：

```
#使用 dd 命令将磁盘 sda 备份到文件/mnt/backup/backup_sda.dd 中
# dd if=/dev/sda of=/mnt/backup/backup_sda.dd
```

上面的示例命令中，dd 命令从设备/dev/sda 中读取数据，并存放到文件 backup_sda.dd 中。

这样磁盘 sda 中的所有数据都被保存到当前目录的 backup_sda.dd 文件内，这些数据包括了磁盘分区表及所有分区和数据等内容。由于磁盘存储的数据可能会有许多，因此命令可能会需要很长时间。

提示：将磁盘备份到文件中时，建议文件名以.dd 结尾，以便于文件使用者识别。

（2）恢复磁盘时，只需要将其读取和写入的文件进行调换即可。例如要将从/dev/sda 中备份的数据恢复到一个新的设备 sdb 中：

```
#使用 dd 命令将备份的数据恢复到另一个新设备中
# dd if=/mnt/backup/backup_sda.dd of=/dev/sdb
```

上面的命令与前面的命令正好相反，dd 命令从文件/mnt/backup/backup_sda.dd 中读取数据，并将这些数据写入磁盘 sdb 中。命令执行完成之后设备 sda 和 sdb 中的数据将完全相同。

（3）许多时候可能希望能够在备份磁盘分区的同时，压缩备份数据以节省空间，这时可以与 gzip 等压缩命令一起使用。

例如备份磁盘 sda 并使用 gzip 压缩备份数据：

```
#将备份数据通过管道传送给 gzip 命令压缩
# dd if=/dev/sda | gzip >/mnt/backup/backup_sda.dd.gz
167772160+0 records in
167772160+0 records out
85899345920 bytes (86 GB) copied, 13131.4 seconds, 6.5 MB/s
```

上面的示例命令中，命令 dd 首先从磁盘 sda 中读取数据，并将这些数据通过管道交给第

2 个命令 gzip。gzip 将这些数据压缩之后，通过重定向的方式写入文件 backup_sda.dd.gz 中。

（4）如果要将使用 gzip 压缩后的数据恢复到磁盘 sdb 中，可以反向使用以上命令：

```
#先使用 gzip 命令解压缩，然后使用 dd 命令将数据恢复到磁盘 sdb 中
# gzip -dc /mnt/backup/backup_sda.dd.gz | dd of=/dev/sdb
```

（5）如果两块磁盘都连接到系统，可以直接将一个磁盘中的内容整盘拷贝到另一个磁盘中，将选项 if 和 of 的参数换成需要使用的参数即可。

例如要将磁盘 sda 中的所有数据拷贝到磁盘 sdb 中：

```
#使用 dd 命令将 sda 中的数据拷贝到 sdb 中
# dd if=/dev/sda of=/dev/sdb
```

（6）利用 dd 命令能直接读写块设备的功能，还可以制作光盘镜像。例如为当前光驱内的光盘制作光盘镜像，并将光盘镜像命名为 linux.iso：

```
#使用 dd 命令从 cdrom 中读取数据，并保存为光盘镜像
# dd if=/dev/cdrom of=/mnt/backup/linux.iso
```

（7）许多时候管理员需要产生一个指定大小的文件，其目的可能是验证一些配置（例如磁盘配额），也可能是测试磁盘、阵列的读写速度等。这时可以使用 dd 命令的 bs 和 count 选项，指定块大小、块数量。

例如产生一个 5GB 的文件放置在当前目录下，并将其命名为 test.5G：

```
#使用选项 bs 指定块大小为 1MB、count 指定块数量为 5120 的方式产生指定大小的文件
# dd if=/dev/zero of=test.5G bs=1M count=5120
5000+0 records in
5000+0 records out
5242880000 bytes (5.2 GB) copied, 56.9074 seconds, 94.3 MB/s
```

dd 命令执行完成后，会显示写入数据花费的时间及写入的平均速度等供使用者参考。管理员也可以使用 dd 命令的此用法评估磁盘性能。

注意： 在上面的命令中用到了一个设备文件/dev/zero，这是一个非常特殊的设备文件，zero 设备的功能是无限制地提供数据 0。

（8）许多时候，磁盘上存放了相当机密的数据，当磁盘报废或者用作其他用途时，删除数据和格式化磁盘都无法保证这些机密数据不被泄露。此时可以使用随机数据填充磁盘的方式销毁磁盘上的数据。

例如要销毁磁盘 sdc 上的数据：

```
#使用随机数设备产生的数据销毁磁盘上的数据
# dd if=/dev/urandom of=/dev/sdc
```

由于随机数设备/dev/urandom 产生的数据比较慢（大概几兆字节每秒），因此上面这条命令需要花费许多时间。

（9）在 Linux 系统中，要备份磁盘的分区表，可以不用其他工具，使用 dd 命令就可以轻松地完成这个工作：

```
#使用 dd 命令备份磁盘 sda 的前 512 个字节
```

```
# dd if=/dev/sda of=image count=1 bs=512
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.00298176 seconds, 172 kB/s
```

上面这个示例命令，会备份磁盘/dev/sda 的前 512 个字节（分区表保存在磁盘的前 512 个字节中），保存到当前目录下并命名为 image。

（10）要从已经备份的文件 image 中恢复分区表：

```
#使用 dd 命令恢复分区表
# dd if=image of=/dev/sda
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.000244524 seconds, 2.1 MB/s
```

dd 命令除了可以对磁盘进行备份、恢复、整盘复制以外，还可以备份、恢复磁盘分区和进行整分区复制等。此处不再赘述，读者可以自行验证。

虽然 dd 命令在备份数据时允许硬盘正在使用，但通常应该停止使用磁盘或以只读方式挂载硬盘，以免出现文件不一致的情况。

9.5　RPM 包管理命令 rpm

RPM 包管理器是由 Red Hat 推出的应用软件管理器(过去的 RPM 就是 Red Hat Package Manager 的缩写）。RPM 包管理器是 Linux 系统中最早的软件包管理软件之一，也是目前最流行的软件包管理器，许多发行版都使用 RPM 作为默认的软件包管理软件。本节将简单介绍 RPM 包管理器及其使用方法。

9.5.1　RPM 包管理器简介

Linux 系统诞生早期，用户安装应用程序需要自行手动编译（编译是利用程序代码生成可执行文件的过程）。手动编译软件虽然有一定优势，但存在安装时间长且需要用户有一定的专业基础等缺点。因此许多发行版推出了软件包管理器，这其中以 RPM 包管理器最为著名，除此之外还有 Debian 使用的 dpkg 等。本小节将简单介绍 Linux 系统中的软件及 RPM 包管理器的特性。

【软件依赖性】

与 Windows 系统中的软件不同，Linux 系统中的软件将不同的功能模块单独写入到不同的软件包中，最后将多个相互依存的软件包结合起来形成一个应用程序。虽然这在 Windows 中也经常使用，但 Linux 系统中的软件模块划分得更小、功能更单一。因此在安装功能性软件包时，通常需要安装许多与其具有依赖关系的软件包，这就是 Linux 系统中的软件依赖性。

【RPM 包管理器】

在早期的系统中，安装应用程序是一件非常麻烦的事情，需要编译许多软件包才完成应用程序的安装。为解决这个问题，Red Hat 公司开发了一个包管理器，早期的包管理器

名为 Red Hat Package Manager。

虽然名称中带有 Red Hat，但这是一个基于 GPL 协议的开源软件。Red Hat 公司鼓励广大 Linux 系统发行厂商了解和使用 RPM 包管理器。由于 Red Hat 的积极推广，现在许多 Linux 发行版都使用 RPM 作为默认的包管理器。现在 RPM 的含义是 RPM Package Manager，由 RPM 社区负责维护。

RPM 为了解决软件包依赖性的问题，在制作软件包的过程中会将软件包的依赖性问题及相关文件信息写入软件包中，以免出现软件包安装后，无法使用或功能不全的问题。当安装软件包时，这些信息将会被 RPM 软件包管理器写入到 RPM 的数据库中，以便于这些软件卸载或更新时能够使用。

【RPM 软件包名称格式】

RPM 软件包名称都有一个固定的格式，以文件服务程序 Samba 的客户端软件包为例，格式为 samba-client-3.0.33-3.7.el5.i386.rpm。这个软件包名称中各部分的含义如下。

- samba-client：最前面的为软件包的名称。
- 3.0.33-3.7：3.0.33 表示软件包的主要版本号，紧跟在后面的 3.7 表示软件发布的次要版本号。
- el5：表示适用于 Red Hat Enterprise Linux 5 系统。fc6 表示适用于 Fedora Core Linux 6 系统。
- i386：表示硬件平台，除此之外还有 i586、i686 等。有些软件包的硬件平台为 noarch，表示该软件包没有硬件平台限制（通常这类软件包中包含的是脚本、说明文档等）。
- .rpm：表示这是一个 RPM 软件包。有些软件包最后是 src.rpm，表示软件包内含有源代码。

RPM 包管理器为软件安装提供了一种比较快捷、方便的方式，让使用者在安装软件包时，不必花费较长的时间编译安装软件的源代码，也从一定程度上解决了软件包依赖性复杂的问题。

9.5.2　rpm 命令基本格式

在 Linux 系统中，可以使用命令 rpm 调用 RPM 包管理器。 rpm 是一个非常复杂的命令，复杂之处在于它拥有众多的选项和参数。本小节将简单介绍 rpm 命令的基本格式。

【命令格式】

```
rpm [OPTION...]
```

【常用选项】

- q：使用查询模式。
- a：查询所有软件包。
- i：显示详细信息。如果指定了软件包，则安装软件包。
- l：显示软件包的文件列表。
- p：查询指定的软件包。
- f：查询指定文件所属软件包。
- v：显示命令执行过程。

- h：安装软件包时显示进度信息。
- e：卸载指定的软件包。
- U：升级软件包。
- force：长格式选项，强制操作，忽略操作过程中的冲突。
- nodeps：长格式选项，忽略操作过程中的软件依赖性，强制操作。此参数可能会导致软件无法使用，应该慎重。
- oldpackage：忽略冲突，强制升级软件包。

在上面的选项中，显示详细信息选项 i 对应的长格式为 info，而安装软件包时，其对应长格式为 install。这两个长格式选项对应的短格式选项都是 i，使用时需要注意区别。

除此之外，rpm 命令还有许多选项，此处不再一一赘述，读者可以阅读相关文档了解和使用。

9.5.3　使用 rpm 命令查询软件包

安装卸载软件时，都需要查询软件包。查询的内容包括查询软件包是否已经安装到系统中、软件包中包含的文件列表等。查询软件包需要使用查询模式选项 q。本小节将简单介绍如何使用 rpm 命令查询软件包。

1．查询已安装的软件

许多时候需要查询系统中已经安装的软件包，例如验证系统中某个软件包版本是否正确、系统中是否已安装某个软件包等。

（1）例如要查询系统中是否装有一个名为 samba-client 的软件包：

```
#使用选项 q 和软件名包称，查询系统中是否安装有指定的软件包
# rpm -q samba-client
samba-client-3.0.33-3.7.el5
```

上面的命令输出表明，当前系统已经安装了 samba-client 软件包，其版本是 3.0.33-3.7.el5。

提示：使用选项 q 加软件包名称的方式查询时，必须使用软件包全名，否则将无法查询到任何结果。

（2）有时可能会忘记某个软件包的具体名称。此时可以使用选项 a 显示系统中已经安装的所有软件包。

查询系统上已经安装的所有软件包：

```
#使用选项 q 和 a 显示系统中已经安装的软件包
# rpm -qa
tzdata-2008i-1.el5
rmt-0.4b41-2.fc6
gnome-mime-data-2.4.2-3.1
……
```

（3）为了能够使用模糊查询，可以在查询全部软件包的同时，配合使用 grep 命令模糊

查询。例如查询当前系统中软件名称包含 ssh 的软件包：

```
#使用管道和 grep 命令模糊查询软件包
# rpm -qa | grep ssh
openssh-4.3p2-29.el5
openssh-server-4.3p2-29.el5
openssh-clients-4.3p2-29.el5
openssh-askpass-4.3p2-29.el5
```

（4）许多时候可能需要查看软件包的详细信息。这时可以配合使用选项 i，显示指定软件包的详细信息。例如查询软件包 chkconfig 的详细信息：

```
#使用选项 i 显示软件包的概况
#软件包概况包括软件版本、安装时间、授权协议、软件功能等信息
# rpm -qi chkconfig
Name        : chkconfig                    Relocations: (not relocatable)
Version     : 1.3.30.1                          Vendor: Red Hat, Inc.
Release     : 2                            Build Date: Wed 14 Nov 2007 01:14:06
AM CST
Install Date: Wed 08 Sep 2010 10:33:42 PM CST   Build Host: hs20-bc1-
6.build.redhat.com
……
```

2. 查询软件包中的文件

有时用户可能需要查询某个软件中包含的文件信息，例如不小心删除一个文件，需要查询并重新安装软件包等。查询软件包中的文件主要使用选项 l 和 f。

（1）查看软件包 chkconfig 中包含的详细文件列表：

```
#使用选项 q 和 l 查看软件包含的文件列表
# rpm -ql chkconfig
/etc/alternatives
/etc/init.d
/etc/rc.d/init.d
/etc/rc.d/rc0.d
……
```

（2）如果需要查看文件列表的不是一个已经安装的软件包，而是一个软件包的安装文件，这时可以使用选项 p 指定输入的参数为一个软件包文件。例如：

```
#使用选项 p 指定输入的参数为一个软件包
# rpm -qpl samba-client-3.0.33-3.7.el5.i386.rpm
warning: samba-client-3.0.33-3.7.el5.i386.rpm: Header V3 DSA signature:
NOKEY, key ID 37017186
/sbin/mount.cifs
/sbin/umount.cifs
/usr/bin/eventlogadm
……
```

（3）也可以使用选项 p 和 i 查看软件包的概况：

```
#使用选项 p 和 i 查看 samba-client 软件包的概况
# rpm -qpi samba-client-3.0.33-3.7.el5.i386.rpm
```

```
warning: samba-client-3.0.33-3.7.el5.i386.rpm: Header V3 DSA signature:
NOKEY, key ID 37017186
Name        : samba-client                Relocations: (not relocatable)
Version     : 3.0.33                          Vendor: Red Hat, Inc.
……
```

（4）有时也可能会需要反向查询一个文件对应的软件包，这种情况可能是无意中删除了一个文件，需要知道应该重新安装的软件包的名称。这时可以使用选项 f 指定要查询的文件。例如：

```
#使用选项 f 指定查询/bin/ls 所属的软件包
#使用选项 f 时应该输入文件绝对路径
# rpm -qf /bin/ls
coreutils-5.97-19.el5
```

此时就可以重新安装命令输出的软件包，恢复删除的文件。

9.5.4　使用 rpm 命令安装软件包

rpm 命令最大的功能是安装软件包，软件包可以从原系统的安装光盘中获取，也可以从互联网上下载。本小节将简单介绍如何使用 rpm 命令安装软件包。

注意：安装下载的软件包时，应该注意软件包对应版本和硬件平台，否则可能会出现无法安装或安装后无法正常使用的情况。

【获取软件包】

获取软件包有两种方法，第 1 种方法是 RHEL5.3 的光盘中自带的软件包，通常位于光盘根目录的 Server 子目录中。第 2 种方法是从互联网下载软件包。现在许多软件都提供了 RPM 安装包，通常可以从软件的官方网站下载。

如果手边没有现成的安装光盘，也不知道软件的官方网站，可以尝试在 Repoforge.org 的软件仓库中查找。

Repoforge 软件包仓库：http://pkgs.repoforge.org/

Repoforge.org 是一个专业提供软件包、源码的下载站点，其收纳了许多软件的安装包和源码。如果找不到需要的软件包，可以尝试在该网站中搜索。

从光盘中获取软件包时，由于光盘中软件包众多，非常不方便查找。此时可以结合使用查看文件列表命令 ls 和 grep 命令，以便于在目录中模糊查找软件包文件。

例如在安装光盘中模糊查找软件包：

```
#使用 ls 和 grep 命令查找名称中包含 ssh 的软件包
# ls /media/Server/ | grep ssh
openssh-4.3p2-29.el5.i386.rpm
openssh-askpass-4.3p2-29.el5.i386.rpm
openssh-clients-4.3p2-29.el5.i386.rpm
openssh-server-4.3p2-29.el5.i386.rpm
```

【安装软件包示例】

安装 RPM 软件包需要使用选项 i（长格式为 install，注意与软件信息选项 info 的区别）及软件包作为参数。

（1）例如需要安装一个软件包：

```
#使用选项 i 安装软件包 cabextract
# rpm -i cabextract-1.3-1.i386.rpm
```

在上面这个示例命令中，用户将不会得到任何提示信息，rpm 将其安装好之后会直接返回命令提示符界面。

（2）如果需要显示安装过程中的详细信息，可以加上选项 v 和 h。例如：

```
#使用选项 v 和 h 显示软件包的详细信息、安装进度等
# rpm -ivh cabextract-1.3-1.i386.rpm
Preparing...          ########################################### [100%]
   1:cabextract       ########################################### [100%]
```

rpm 命令在安装软件包的同时，还输出了安装进度等信息。

（3）然而并不是所有软件包的安装过程都这么顺利，许多安装包会遇到依赖性问题，即在安装应用程序之前，应该先安装能让该应用程序正常工作的前提软件包。

如果遇到软件包存在依赖性问题，应该先安装软件所依赖的前提软件包，或在同一条命令中同时安装前提软件包。

如果安装的软件包依赖于另一个软件包，rpm 命令会给出相应的提示。例如要安装 gcc 编译器（gcc 是 Linux 等类 UNIX 系统中主要的软件编译器，它是一个开源软件）：

```
#使用 rpm 命令安装 gcc 编译器
# rpm -ivh gcc-4.1.2-44.el5.i386.rpm
warning: gcc-4.1.2-44.el5.i386.rpm: Header V3 DSA signature: NOKEY, key ID
37017186
#以下为依赖信息详情
#gcc 编译器依赖于两个软件包：glibc-devel 和 libgomp
#两个软件包的版本要求大于或等于 2.2.90-12 和 4.1.2-44
#软件包 libgomp 的版本限制为 el5
error: Failed dependencies:
       glibc-devel >= 2.2.90-12 is needed by gcc-4.1.2-44.el5.i386
       libgomp >= 4.1.2-44.el5 is needed by gcc-4.1.2-44.el5.i386
```

rpm 命令在安装 gcc 软件包时遇到了错误，命令输出了其依赖的两个包：glibc-devel 和 libgomp。

从光盘中找到并安装软件包 glibc-devel 和 libgomp 时，又需要安装 glibc-headers 和 kernel-headers 这两个互相依赖的软件包。

为了解决复杂的依赖关系，可以将这些软件包都作为 rpm 命令的参数一并安装：

```
#将多个依赖的软件包都作为 rpm 命令的参数，一并安装
# rpm -ivh gcc-4.1.2-44.el5.i386.rpm glibc-devel-2.5-34.i386.rpm libgomp-
4.3.2-7.el5.i386.rpm glibc-headers-2.5-34.i386.rpm kernel-headers-2.6.18-
128.el5.i386.rpm
warning: gcc-4.1.2-44.el5.i386.rpm: Header V3 DSA signature: NOKEY, key ID
37017186
Preparing...          ########################################### [100%]
   1:libgomp          ########################################### [ 20%]
   2:kernel-headers   ########################################### [ 40%]
   3:glibc-headers    ########################################### [ 60%]
```

```
  4:glibc-devel        ########################################### [ 80%]
  5:gcc                ########################################### [100%]
```

从上面的示例命令中可以看到 rpm 命令先计算软件包的依赖性，然后依次安装 gcc 编译器依赖的软件包，最后完成 gcc 编译器的安装工作。

在同一条命令中安装多个依赖包的方法，同样适用于几个软件包之间相互依赖的情况（例如 A 依赖于 B，B 又依赖于 A 等）。

（4）有时管理员可能需要忽略软件包的依赖性，强制安装软件包。此时可以使用选项 nodeps 忽略软件的依赖性，这个选项适用于依赖性错误的情况，例如有些软件包可能会依赖软件包本身。

有时误删除一个文件，需要重新安装时，rpm 命令会提示文件冲突。此时可以配合选项 force 忽略软件包、文件冲突，强制安装软件包。

例如强制安装软件包 samba-client：

```
#使用选项 force 和 nodeps 强制安装 samba-client 软件包
# rpm -ivh --force --nodeps samba-client-3.0.33-3.7.el5.i386.rpm
warning: samba-client-3.0.33-3.7.el5.i386.rpm: Header V3 DSA signature:
NOKEY, key ID 37017186
Preparing...           ########################################### [100%]
  1:samba-client       ########################################### [100%]
```

提示： 忽略文件的依赖性、强制安装都有可能令安装的软件在运行中出现问题或无法使用。因此除非目的明确，否则不建议使用这种方式安装软件包。

9.5.5　使用 rpm 命令卸载软件包

软件包管理的又一个任务是卸载不再使用的软件包。卸载需要使用选项 e 及软件包的全名，因此在使用 rpm 卸载软件时，应该先使用选项 q 查询需要卸载软件的全名。本小节将简单介绍如何使用 rpm 命令卸载软件包。

（1）如果知道需要卸载的软件包名称，直接使用选项 e 卸载：

```
#使用选项 e 卸载软件包 samba-client
# rpm -e samba-client
```

命令会卸载指定的软件包，然后直接返回到提示符界面。

（2）卸载软件时可能也会出现依赖性问题，例如：

```
#使用选项 e 卸载软件包时提示软件依赖性
# rpm -e kernel-headers
#以下是依赖性提示
#rpm 命令提示有一个已安装的、名为 glibc-headers 的软件依赖于当前卸载的软件包
error: Failed dependencies:
       kernel-headers is needed by (installed) glibc-headers-2.5-34.i386
       kernel-headers >= 2.2.1 is needed by (installed) glibc-headers-2.5-
34.i386
```

从上面的示例命令输出中可以看到，另一个软件包 glibc-headers 依赖于当前需要卸载的软件包。如果确定不再使用这些软件包，可以按照其依赖性提示，将这些依赖的软件包

一一卸载。

（3）如果要卸载的软件包关系复杂，也可以同时卸载所有依赖的软件包：

```
#使用选项 e 同时卸载多个依赖的软件包
# rpm -e kernel-headers glibc-headers glibc-devel gcc
```

使用 rpm 命令安装、卸载软件包时，需要特别注意软件的依赖性。强制安装、卸载都有可能会导致软件不完整，影响软件的正常使用。

9.5.6　使用 rpm 命令升级软件包

许多时候，管理员需要升级系统中已经安装的软件包，即将系统中的软件从当前版本升级到更高的版本。高版本的软件可能会附带新的功能，消除旧版本存在的漏洞等。本小节将简单介绍如何使用 rpm 命令升级软件包。

使用 rpm 命令升级软件包需要使用选项 U。也可以同时使用选项 v 和 h，显示升级过程中的详细信息、升级进度等内容。

（1）例如使用选项 U 升级软件包：

```
#使用选项 U 升级软件 bzip2-libs
# rpm -Uvh bzip2-libs-1.0.3-4.el5_2.i386.rpm
```

（2）由于系统中安装有低版本的软件，因此在安装过程中可能会出现冲突，导致升级失败，此时可以使用选项 oldpackage 强制升级：

```
#使用选项 oldpackage 强制升级软件包
# rpm -Uvh --oldpackage bzip2-libs-1.0.3-4.el5_2.i386.rpm
```

9.6　编译安装相关命令和工具

Linux 系统诞生早期，许多应用程序都是通过编译安装的。虽然使用编译安装方式比较费时，但也存在许多优点。

- ❑ 可以获得更新的软件（软件更新时，可能还没来得及制作软件包）。
- ❑ 编译安装的软件可以按需要定制。编译安装时，可以通过开启、禁用某些功能获得更好的性能。
- ❑ 编译安装的软件会按系统硬件的实际情况编译某些模块，因此拥有更好的适用性。
- ❑ 高级用户可以按需要修改源代码，为自己量身定制软件。

本节将以媒体播放软件 Mplayer 为例，介绍如何编译安装应用程序。

9.6.1　安装编译环境

在编译安装应用之前，首先需要安装编译环境。Linux 系统中的大多数软件使用的编译环境都是 gcc，因此应该先安装 gcc 编译环境。如果软件需要使用图形界面，可能还需

要安装 gtk+编译环境（gtk+是 Linux 系统中的图形编译环境，与 gcc 一样，是一个开源软件）。本小节将简单介绍如何安装编译环境。

（1）可以使用 rpm 命令检查系统是否已经安装了 gcc 编译环境：

```
#使用 rpm 命令检查系统中是否已经安装了 gcc 编译环境
# rpm -qa | grep gcc
compat-libgcc-296-2.96-138
gcc-4.1.2-44.el5
libgcc-4.1.2-44.el5
```

如果系统中没有安装编译环境，可以挂载光驱，然后在光驱的安装包目录中使用以下命令：

```
#使用 rpm 命令安装编译环境
# rpm -ivh gcc-4.1.2-44.el5.i386.rpm libgomp-4.3.2-7.el5.i386.rpm glibc-devel-2.5-34.i386.rpm  glibc-headers-2.5-34.i386.rpm  kernel-headers-2.6.18-128.el5.i386.rpm
warning: gcc-4.1.2-44.el5.i386.rpm: Header V3 DSA signature: NOKEY, key ID 37017186
Preparing...           ########################################### [100%]
   1:libgomp           ########################################### [ 20%]
   2:kernel-headers    ########################################### [ 40%]
……
```

编译安装某些软件时，可能还需要其他前提软件或编译环境。可以阅读软件的说明，按提示安装需要的软件和特殊的编译环境。

（2）安装图形界面中的软件时，需要安装 gtk+编译环境。可以在光盘的安装程序目录中使用以下命令安装 gtk+软件包：

```
#在光盘软件包目录中使用以下命令安装 gtk+编译环境
# rpm -ivh zlib-1.2.3-3.i386.rpm
# rpm -ivh zlib-devel-1.2.3-3.i386.rpm
# rpm -ivh libpng-devel-1.2.10-7.1.el5_0.1.i386.rpm
#  rpm  -ivh  gtk+-1.2.10-56.el5.i386.rpm  gdk-pixbuf-0.22.0-25.el5.i386.rpm glib-1.2.10-20.el5.i386.rpm
# rpm -ivh gtk+-devel-1.2.10-56.el5.i386.rpm glib-devel-1.2.10-20.el5.i386.rpm libX11-devel-1.0.3-9.el5.i386.rpm libXext-devel-1.0.1-2.1.i386.rpm libXi-devel-1.0.1-3.1.i386.rpm libXau-devel-1.0.1-3.1.i386.rpm libXdmcp-devel-1.0.1-2.1.i386.rpm xorg-x11-proto-devel-7.1-9.fc6.i386.rpm mesa-libGL-devel-6.5.1-7.7.el5.i386.rpm
```

从本小节中的编译环境安装过程可以看出，RPM 软件包之间依赖关系的复杂性。

注意： *在多用户系统中安装编译环境可能会引发安全性问题，例如未授权用户使用编译环境在系统中安装非法软件等，因此应该特别注意。*

9.6.2 获取软件工具 wget、links

互联网上有许多开放源码软件，这些软件可以为用户完成各方面的应用需要。许多初学者可能不知道如何获取这些软件，这时可以从互联网的软件下载站中了解这些软件。下

面是国内比较有影响力的一些 Linux 软件下载站点：

- China UNIX 软件下载中心：http://download.chinaunix.net/disc/linux/
- 中国 IT 实验室下载中心：http://download.chinaitlab.com/
- 红联 Linux 门户：http://www.linuxdiyf.com/
- 华军软件园 Linux 下载频道：http://linux.newhua.com/

如果要下载软件的最新版，通常可以在谷歌（http://www.google.com/）搜索其官方网站下载。

1．使用wget工具下载软件

如果在远程终端中操作，可以使用 wget 工具下载已知网址的软件（在本地操作系统中复制软件的网址，然后在 Putty 等终端中单击鼠标右键粘贴网址）。

wget 是 Linux 系统中常用的下载工具，它可以使用 HTTP、FTP 等多种协议。在使用 wget 下载软件之前，应该先设置系统网络（设置网络相关内容将在第 10 章中介绍）。

【命令格式】

```
wget [option] <URL>
```

【常用选项】

最常用的选项只有 c，该选项的功能是续传上次没有下载完成的任务。使用选项 c 的前提条件是当前工作目录中保存有上次没有下载完成的文件。

【用法示例】

（1）获取到软件的网址后，就可以从相应的网站下载要安装的软件了。从 Mplayer 站点上下载其源码：

```
#使用 wget 命令下载指定的 URL
#命令将在下载完成后自动退出，并将下载的文件保存在当前目录中
# wget http://www.mplayerhq.hu/MPlayer/releases/MPlayer-1.0rc4.tar.bz2
--07:46:52--
http://www.mplayerhq.hu/MPlayer/releases/MPlayer-1.0rc4.tar.bz2
Resolving    www.mplayerhq.hu...    193.225.187.202,    199.125.85.41,
213.144.138.186, ...
Connecting to www.mplayerhq.hu|193.225.187.202|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9650074 (9.2M) [application/x-bzip2]
Saving to: `MPlayer-1.0rc4.tar.bz2'

100%[=======================================>] 9,650,074   604K/s  in 27s

07:47:19 (355 KB/s) - `MPlayer-1.0rc4.tar.bz2' saved [9650074/9650074]
```

（2）如果下载过程被中断，可以使用选项 c 续传未完成的任务：

```
#使用选项 c 续传未完成的任务
#此用法只能在包含未完成任务文件的目录中执行
#此命令需要目标网站支持续传功能
# wget -c http://www.mplayerhq.hu/MPlayer/releases/MPlayer-1.0rc4.tar.bz2
```

使用以上命令后，wget 命令会自动续传未完成的任务。

2. 字符界面中的浏览器links

如果要在字符界面中下载软件，可以使用字符界面中最简单的浏览器 links。直接使用命令 links 和网站的网址即可启动 links 浏览器：

```
#使用 links 浏览器打开指定的网站
# links http://www.mplayerhq.hu
```

此时系统会立即启动 links 浏览器并打开网站，如图 9.1 所示。

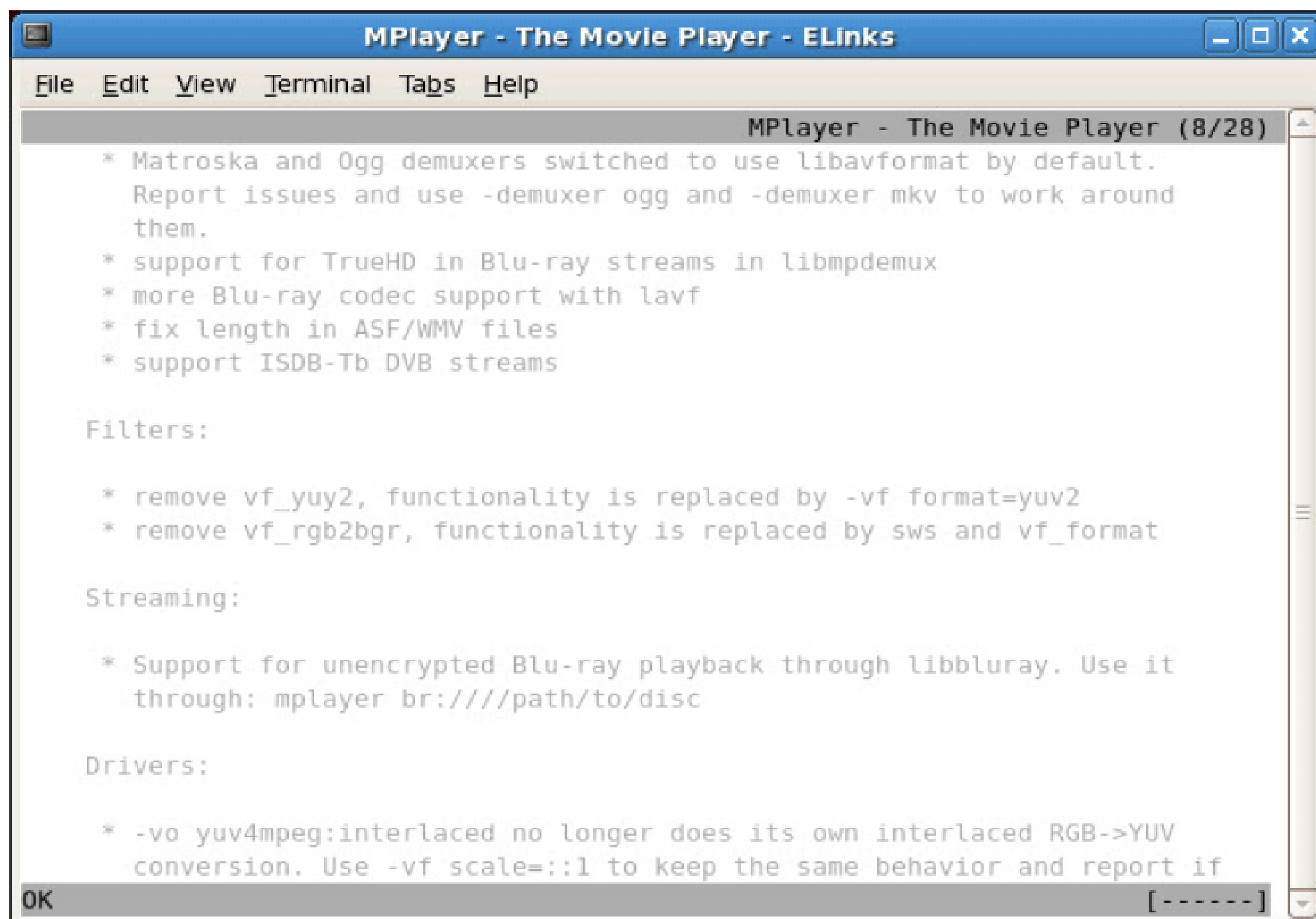


图 9.1　links 浏览器

在 links 浏览器中，可以使用上、下方向键切换选择要操作的超链接，跳转到超链接可以按下 Enter 键。如果要查看其菜单，可以按 F9 键；退出 links 浏览器，可以按 q 键。

由于 links 浏览器用法非常简单，此处不再赘述，感兴趣的读者可以阅读相关文档。

除了本节中介绍的方法以外，读者可以使用熟悉的工具下载软件，使用 WinSCP 上传到 Linux 系统中。也可以使用 Linux 系统的图形化浏览器，例如 Firefox 等。

9.6.3　编译前的配置

在编译安装之前，应该先执行配置工作。配置工作需要使用软件源码目录中自带的可执行文件 configue。本小节将简单介绍如何配置源码选项。

注意：开始安装软件前，应该先查看软件源码中的说明文件（通常其名称为 README）。说明文件中可能会列出需要安装的前提软件、编译时需要注意的事项（例如有些软件不允许并行编译）等。

（1）下载的源码包通常都会使用 tar 工具归档，首先需要从下载的归档文件中恢复源码目录：

```
#使用 tar 命令恢复下载的归档文件包
# tar -jxvf MPlayer-1.0rc4.tar.bz2
MPlayer-1.0rc4/
MPlayer-1.0rc4/debian/
MPlayer-1.0rc4/debian/control
MPlayer-1.0rc4/debian/dirs
……
```

tar 命令将恢复的文件放到当前目录中的 MPlayer-1.0rc4 子目录中。

（2）恢复归档文件后，就可以进入源代码目录，查看配置的帮助信息：

```
#使用 cd 命令进入源码目录
# cd MPlayer-1.0rc4
#执行源码目录中的 configure，并使用 help 选项查看帮助
#不同的软件配置命令的选项和功能都不同，因此编译安装前都应该使用以下命令查看帮助
# ./configure --help
Usage: ./configure [OPTIONS]...

Configuration:
  -h, --help            display this help and exit

Installation directories:
  --prefix=DIR          prefix directory for installation [/usr/local]
  --bindir=DIR          directory for installing binaries [PREFIX/bin]
  --datadir=DIR         directory for installing machine independent
                        data files (skins, etc) [PREFIX/share/mplayer]
……
```

使用可执行文件 configure 配置软件时，常用的选项为 prefix，该选项用于指定程序的安装目录。

（3）如果不使用 prefix 指定程序的安装目录，程序会使用/usr/local 作为其安装目录。用户可以指定选项 prefix 的值自定义应用程序的安装目录。

本例将程序安装到目录/usr/local/MPlayer 中：

```
#由于配置文件不会自动生成目录，因此需要手动生成安装目录
# mkdir -p /usr/local/MPlayer
#使用选项 prefix 指定程序的安装目录
# ./configure --prefix=/usr/local/MPlayer
Detected operating system: Linux
Detected host architecture: i386
Checking for cc version ... 4.1.2
Checking for host cc ... cc
Checking for cross compilation ... no
……
```

这个命令会产生大量的输出，而且非常耗时。用户可以从命令输出中，了解配置命令 configure 检查的系统项。

在配置过程中，configure 会检查系统是否符合软件的运行环境，例如内核支持、硬件支持、相关库文件支持、编译环境等。如果软件需要其他软件的支持，也会一并检查并给出提示。

在配置过程中，用户应该特别注意被标记为错误的检查结果（错误往往是由于软件需要某种支持或前提软件缺失等，这些都会导致软件安装失败），对标记为 no 的检查结果通常可以忽略。

（4）当然可能许多人会认为终端模式使用起来十分不方便，需要使用 MPlayer 的图形界面。可以在配置时加上选项 enable-gui：

```
#使用选项 enable-gui 开启图形界面运行
# ./configure --enable-gui --prefix=/usr/local/MPlayer
```

如果配置过程没有出现任何错误，接下来就可以编译源代码了。

9.6.4　编译软件命令 make

使用可执行文件 configure 完成配置工作后，就需要使用命令 make 对软件执行编译了。在编译软件过程中，gcc 编译器会利用软件的源码，为软件生成应用程序运行时必需的可执行文件、共享库文件。

在软件的源码目录中对应用程序执行编译：

```
#使用 make 命令编译软件源码
# make
help/help_create.sh help/help_mp-en.h UTF-8
cc -O -DCODECS2HTML -I. -o codec-cfg codec-cfg.c
./codec-cfg etc/codecs.conf > codecs.conf.h
./version.sh `cc -dumpversion`
……
```

通常这个过程消耗的时间比 configure 命令消耗的时间要长得多，并且会产生许多输出。

如果在软件编译过程中出现错误（通常是由于某个支持的软件或必需的头文件没有找到引起的），可能需要查阅相关说明文档并重新配置，然后再编译。

提示：如果软件需要进行二次编译，建议在二次编译前，先使用 make clean 命令清除上次编译生成的文件。

9.6.5　安装命令 make install

编译成功之后，就可以使用命令 make install 安装应用程序了。在安装过程中，make install 会使用选项 prefix 指定的目录，将应用程序的库文件、可执行文件、帮助文档等安装到指定的目录中。除此之外，有些应用程序可能还会添加相关支持、生成配置文件、设置文件权限等。

（1）在本例中可以直接使用 make install 安装软件：

```
#使用 make install 安装编译好的软件
# make install
install -d /usr/local/bin /usr/local/etc/mplayer /usr/local/lib
install -m 755 -s mencoder /usr/local/bin
install -d /usr/local/share/man/man1
install -m 644 DOCS/man/en/mplayer.1 /usr/local/share/man/man1/
```

```
cd /usr/local/share/man/man1 && ln -sf mplayer.1 mencoder.1
install -m 755 -s mplayer /usr/local/bin
```

从上面的命令输出中可以看到，安装过程将应用程序要使用的程序文件、库文件等放入相应的目录中。这些目录包括 bin、lib、man、share 等，分别用于存放程序文件、库文件、帮助手册文件。

（2）为了运行图形化的 MPlayer，还需要为其安装默认的皮肤。如果在安装时选择了 MPlayer 使用的语言，还需要为其安装相关的字体文件。

下载皮肤文件：

```
#使用 wget 命令下载 MPlayer 的皮肤
# wget http://www3.mplayerhq.hu/MPlayer/skins/Blue-1.7.tar.bz2
```

下载完成之后，将其解压缩：

```
#使用 tar 命令解压皮肤文件
# tar -jxvf Blue-1.7.tar.bz2
Blue/
Blue/main-silver.png
……
```

将皮肤文件复制到软件包中的皮肤目录并重命名：

```
#使用 cp 命令安装皮肤文件
# cp -r Blue /usr/local/MPlayer/share/mplayer/skins/default
```

9.6.6　运行及环境配置

安装软件的目的是运行软件，在本小节中将介绍如何运行已经安装好的软件、配置软件的运行环境等内容。

1. 运行软件

软件被安装在了目录/usr/local/MPlayer 中，可以首先查看该目录下的文件：

```
#使用 ls 命令查看安装目录中的子目录
# ls -l /usr/local/MPlayer/
total 32
drwxr-xr-x 2 root root 4096 Feb  3 15:25 bin
drwxr-xr-x 3 root root 4096 Feb  3 15:25 etc
drwxr-xr-x 2 root root 4096 Feb  3 15:25 lib
drwxr-xr-x 6 root root 4096 Feb  3 15:25 share
```

通常程序文件会放在 bin 目录中，可以查看该目录中的文件并运行。

（1）由于软件的相关设置没有写入环境变量，因此需要使用绝对路径的方式运行软件：

```
#使用绝对路径的方式运行 MPlayer，并打开文件 5.mp3
#如果没有使用 prefix 选项自定义安装目录，可以直接使用命令 mplayer，不必加上全路径
# /usr/local/MPlayer/bin/mplayer 5.mp3
#以下为 MPlayer 播放器在字符界面中的交互式界面
MPlayer 1.0rc4-4.1.2 (C) 2000-2010 MPlayer Team
```

```
Playing 5.mp3.
Audio only file format detected.
Clip info:
 Title:
 Artist:
 Album:
 Year: 2009
 Comment:
 Genre: Other
==========================================================================
Opening audio decoder: [mp3lib] MPEG layer-2, layer-3
AUDIO: 44100 Hz, 2 ch, s16le, 320.0 kbit/22.68% (ratio: 40000->176400)
Selected audio codec: [mp3] afm: mp3lib (mp3lib MPEG layer-2, layer-3)
==========================================================================
AO: [oss] 44100Hz 2ch s16le (2 bytes per sample)
Video: no video
Starting playback...
A:   7.8 (07.8) of 209.0 (03:29.0)  1.1%
```

上面的示例命令使用 MPlayer 播放名为 5.mp3 的音乐文件。如果要退出正在字符界面中运行的 MPlayer，可以按 q 键。

（2）如果安装了图形化的 MPlayer，可以在图形界面中运行以下命令启动 MPlayer：

```
#使用绝对路径的方式运行图形化的 MPlayer
# /usr/local/MPlayer/bin/gmplayer
```

图形化的 MPlayer 如图 9.2 所示。



图 9.2　图形化的 MPlayer

2．环境配置

细心的读者可能已经发现，安装的软件不能像运行其他软件和命令那样，直接输入命令运行。在上面的例子中，均使用了绝对路径的方式运行软件。这是因为新安装的软件使用了自定义路径（如果没有自定义软件的安装目录，可跳过环境配置步骤），这个路径并

不在系统环境变量 PATH 中，为了可以更快捷地运行软件，还需要配置环境变量。

（1）在命令行中配置环境变量，可以使用如下命令：

```
#将 MPlayer 的程序目录保存到环境变量 PATH
# PATH=$PATH:/usr/local/MPlayer/bin
#使用 export 将 PATH 变量定义为全局变量
# export PATH
```

上面的命令的作用是将软件的程序目录加到变量 PATH 结尾，此时就可以像使用命令那样直接运行 mplayer 和 gmplayer 了。

（2）但使用上面的方法将会在系统重新启动后失效。如果需要使其在重启后仍然有效，可以使用修改系统配置文件的方法。

将环境变量保存到配置文件中时，如果仅需要为当前用户添加环境支持，可以将相关配置语句写入文件~/.bash_profile 中：

```
#使用 echo 命令将相关语句写入当前用户的配置文件
#注意以下使用的是>>（追加方式写入文件）
# echo "#add MPlayer directory">>~/.bash_profile
# echo "PATH=\$PATH:/usr/local/MPlayer/bin" >>~/.bash_profile
# echo "export PATH" >>~/.bash_profile
```

如果需要为所有用户添加环境支持，可以使用以下命令：

```
#使用 echo 命令将环境变量语句写入全局用户文件中
# echo "#add MPlayer directory">>/etc/profile
# echo "PATH=\$PATH:/usr/local/MPlayer/bin" >>/etc/profile
# echo "export PATH" >>/etc/profile
```

上述命令均是将相关命令写入到相关的配置文件中，重新启动或重新登录系统后即可生效，关于这些文件的详细使用方式和说明将在第 16 章中介绍。

9.6.7　卸载软件命令 make uninstall

使用编译安装的软件通常可以使用 make uninstall 命令卸载，此命令只能在软件源码目录中使用：

```
#使用 make uninstall 命令卸载编译安装的软件
# make uninstall
```

以上命令需要源码目录中的相关文件支持（主要是源码目录中的 Makefile 等文件），如果文件不支持，就无法使用以上命令卸载软件。

对于不能使用 make uninstall 命令卸载的软件，如果使用了自定义软件安装路径的方式安装软件，只需要将环境配置、安装目录及其中的所有文件删除即可。如果使用默认路径，则不能删除（可能会删除掉其他软件或相关帮助信息等）。

9.7　利用 yum 工具安装应用程序

从上一节安装 gtk+编译环境的例子中可以看出，RPM 软件包之间的依赖关系非常复

杂。在实际管理过程中，这种依赖关系可能会更加复杂。因此非常有必要寻找一种自动化安装工具，让安装工具自己处理这些关系复杂的依赖关系。幸运的是 Red Hat 推出了一个名为 Yellowdog Updater Modified 的包管理器（通常简称为 yum）。本节将简单介绍如何使用 yum 管理应用程序。

9.7.1　yum 简介

对于用户而言，RPM 软件包管理器为用户提供了一个非常便捷的应用程序安装环境，也在一定程度上解决了依赖性复杂的问题。但有些复杂的依赖性问题仍然会给用户造成不小的麻烦。这些麻烦来自 RPM 软件包管理器给出的让用户无从着手的提示，例如安装 httpd 软件包：

```
#使用 rpm 命令安装 httpd 软件包
# rpm -ivh httpd-2.2.3-22.el5.i386.rpm
warning: httpd-2.2.3-22.el5.i386.rpm: Header V3 DSA signature: NOKEY, key
ID 37017186
#注意以下依赖关系
error: Failed dependencies:
        libapr-1.so.0 is needed by httpd-2.2.3-22.el5.i386
        libaprutil-1.so.0 is needed by httpd-2.2.3-22.el5.i386
```

命令提示用户，当前安装的 httpd 软件包需要两个前提库文件 libapr-1.so.0 和 libprutil-1.so.0，许多用户都不明白这两个库文件来自于哪个软件包。如果要从所有的软件包中遍历查找这两个库文件，其难度和花费的时间可想而知（本书的后续章节中将会介绍遍历库文件的脚本，读者可以自行研究）。

为此 Red Hat 推出了 Yellowdog Updater Modified 包管理器，yum 包管理器可以自行计算要安装软件包的依赖性，并自动安装软件包需要的软件包和库文件。

使用 yum 包管理器时，需要为其设置 yum 使用的软件源（软件源也称为软件仓库，其中包含了若干 rpm 安装包和 yum 工作的索引文件）。对于使用 Fedora Core Linux 发行版的用户而言，可以直接使用操作系统自带的 yum 源。使用 Red Hat Enterprise Linux 发行版的用户，如果未将系统注册到红帽网络，将无法使用官方的源。

9.7.2　配置 yum

yum 可以使用互联网或本地保存的安装包作为源。当用户指定安装某个软件时，yum 将会查找源，并计算软件包的依赖性，然后安装软件包。yum 安装软件包的整个过程都不需要用户参与，大大降低了安装软件包的难度。本小节将介绍如何使用安装光盘作为 yum 的源。

1．yum的配置文件

yum 使用的配置文件为/etc/yum.conf，这个文件是 yum 工作的核心配置文件。查看配置文件的内容如下：

```
#使用 cat 命令查看 yum 的配置文件内容
# cat /etc/yum.conf
[main]
cachedir=/var/cache/yum
keepcache=0
debuglevel=2
……
```

这个配置文件的内容十分简洁，每一行就是一个配置项，其中配置了 yum 的缓存目录、错误级别、日志文件等。通常不需要对这个文件中的内容作特殊的设置。

除了 yum 工作的配置文件外，yum 还有源配置文件。这些文件通常位于目录 /etc/yum.repos.d 中，默认情况下该目录中只有一个配置文件 rhel-debuginfo.repo。

查看 yum 源配置文件的内容：

```
#使用 cat 命令查看 yum 的源配置文件
# cat rhel-debuginfo.repo
[rhel-debuginfo]
name=Red Hat Enterprise Linux $releasever - $basearch - Debug
baseurl=ftp://ftp.redhat.com/pub/redhat/linux/enterprise/$releasever/en
/os/$basearch/Debuginfo/
enabled=0
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

这个配置文件比较简单，每行为一个配置项，并使用等号“=”为每个配置项赋值。这些配置项及其含义如下。

- [name]：软件源的名称。
- name：软件源的描述信息。
- baseurl：软件源的位置。默认的软件源为 Red Hat 官方的软件源。
- enabled：是否需要启用这个软件源，1 为启用，0 为禁用。
- gpgcheck：是否需要验证 GPG 签名。GPG 签名用于验证软件包是否来自 Red Hat 官方。
- gpgkey：GPG 签名的验证密钥信息存放位置。

了解这个文件中各选项的含义之后，就可以使用安装光盘建立 yum 源，然后使用 yum 工具安装软件了。

2．使用DVD光盘配置软件源

RHEL5.3 的安装光盘分为 DVD 和 CD 两种，DVD 安装光盘将所有需要的内容和安装文件都集中放置在一张光盘上，而 CD 安装光盘则分别存放在 4 张光盘内。因此使用的介质不同，操作也有差异。首先将介绍如何使用 DVD 光盘建立 yum 源。

由于 DVD 光盘已经将所有需要的文件集中放置，因此可以直接将 DVD 光盘挂载到系统中，使用光盘建立 yum 源。

（1）在配置 yum 源之前，应该先备份其源配置文件：

```
#使用 cp 命令备份 yum 源配置文件
# cp /etc/yum.repos.d/rhel-debuginfo.repo /etc/yum.repos.d/rhel-
```

```
debuginfo.repo.bak
```

（2）挂载光驱到目录/mnt，以便于指定光驱目录为 yum 源：

```
#将光驱挂载到目录/mnt 上
# mount /dev/cdrom /mnt
mount: block device /dev/cdrom is write-protected, mounting read-only
```

（3）挂载完成后修改 yum 源配置文件，将文件 rhel-debuginfo.repo 的内容修改如下：

```
#使用 cat 命令查看修改后的源配置文件
# cat /etc/yum.repos.d/rhel-debuginfo.repo
[rhel-debuginfo]
name=Red Hat Enterprise Linux $releasever - $basearch - Debug
baseurl=ftp://ftp.redhat.com/pub/redhat/linux/enterprise/$releasever/en
/os/$basearch/Debuginfo/
#保持 enabled 的值为 0
enabled=0
……
#除标识的内容外，以上内容无其他变化

#以下为新写入的内容
#新源的名称为 cdrom
[cdrom]
name=Red Hat DVD
#使用本地目录作为软件源的路径
baseurl=file:///mnt/Server/
#将 enabled 的值设为 1，启用新建立的源
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

新建的软件源中，软件源的目录为/mnt/Server。此处的“file://”与前面的“ftp://”类似，表示使用的协议，官方的源中使用的“ftp://”表示使用 FTP 协议。此处的“file://”表示使用本地文件系统，而其后的/mnt/Server 则表示源目录在本地文件系统上的具体位置。

配置好以上内容之后，使用 DVD 安装光盘建立的 yum 源就可以使用了。

3．使用CD安装光盘配置软件源

使用 CD 安装光盘时，由于安装文件分布在不同的光盘上，因此建立 yum 源之前，需要将 4 张安装光盘内的安装文件都复制到硬盘的同一个目录中，然后再修改 yum 源配置文件。

（1）在本例中，先将所有光盘中的安装文件（即光盘上所有以“.rpm”结尾的文件）都复制至目录/mnt/cd_yum/rpm 内，具体操作此处不赘述。

（2）yum 工作时依赖于几个索引文件，这几个索引文件被放置在光盘上的安装文件目录中的 repodata 子目录中。查看索引文件：

```
#使用 ls 命令查看光盘中的索引文件
#此目录位于第 1 张 CD 光盘中
# ls /media/Server/repodata/
comps-rhel5-server-core.xml  filelists.xml.gz  other.xml.gz  primary.
```

```
xml.gz  repomd.xml  TRANS.TBL
```

上面这个示例命令假定光盘已经挂载到目录/media。命令输出的几个文件 filelists.xml.gz、other.xml.gz、primary.xml.gz 和 repomd.xml，都是 yum 工作时必需的文件。如果在复制时没有复制到这几个文件，或者这几个文件的内容不准确，都会影响 yum 的正常使用。

用户复制安装文件时，可以一并复制这个目录，也可以使用 createrepo 命令重新创建这个目录中的文件。

（3）重新创建索引文件需要使用命令 createrepo，默认情况下，系统并没有安装 createrepo 命令的软件包。可以使用以下命令安装：

```
#使用 rpm 命令安装软件包 createrepo
# rpm -ivh createrepo-0.4.11-3.el5.noarch.rpm
Preparing...            ########################################### [100%]
   1:createrepo         ########################################### [100%]
```

安装完成后就可以建立索引文件了：

```
#使用 createrepo 命令建立索引文件
# createrepo /mnt/cd_yum/rpm/
2255/2255 - opensm-devel-3.2.2-3.el5.i386.rpm                          m
Saving Primary metadata
Saving file lists metadata
Saving other metadata
```

由于命令会搜寻、分析所有软件包，并建立索引文件，因此这个命令会耗费大量时间。命令完成之后，将在目录/mnt/cd_yum/rpm/中建立一个新的子目录 repodata，并将新生成的索引文件放置在其中。

（4）完成以上步骤后，就可以修改 yum 源配置文件了。具体可以参考使用 DVD 建立 yum 源步骤中的配置文件，修改完成后就可以使用 yum 安装软件包了。

注意：使用安装光盘建立 yum 源，仅仅是为了安装软件包时方便，不能为系统更新软件，因此建议使用官方的 yum 源，以便于获取更多更新的软件包。

9.7.3　查询源上的软件包

使用 yum 安装软件包之前，可能需要查询源上的软件包，也可能因不记得软件包的具体名称而需要模糊查询。本小节将简单介绍如何使用 yum 查询软件包。

【命令格式】

```
yum search package_name
```

查询软件需要使用 search 参数，yum 软件会查询源上所有包含 package_name 的软件包，并列出所有查找到的结果。

【用法示例】

例如查询包含 httpd 的软件包：

```
#使用 yum 查看软件包 httpd
# yum search httpd
```

```
#查询所有可以使用的源
Failed to set locale, defaulting to C
Loaded plugins: rhnplugin, security
This system is not registered with RHN.
RHN support will be disabled.
============================ Matched: httpd ============================
#以下为所有查询到的结果
mod ssl.i386 : SSL/TLS module for the Apache HTTP server
system-config-httpd.noarch : Apache configuration tool
httpd.i386 : Apache HTTP Server
httpd-devel.i386 : Development tools for the Apache HTTP server.
httpd-manual.i386 : Documentation for the Apache HTTP server.
mod_dav_svn.i386 : Apache server module for Subversion server.
```

从上面的示例命令中可以看出，yum 列出了所有与查询内容相关的软件包。用户可以从查询的软件结果列表中选择需要使用的软件包。

9.7.4　利用 yum 安装软件包

查找到需要安装的软件包之后，就可以使用 yum 命令安装软件包了。yum 包管理器的使用方法与 RPM 包管理器类似，本小节将简单介绍如何使用 yum 安装软件包。

【命令格式】

```
yum [-y] install soft_package_name
```

使用 yum 命令安装软件包时，可以使用选项 y，该选项将会自动允许 yum 的所有操作而不提示用户。

【用法示例】

（1）例如要安装 Web 服务器程序：

```
#使用 yum 命令安装软件包 httpd
# yum install httpd
#以下为软件源检查过程
Loaded plugins: rhnplugin, security
This system is not registered with RHN.
RHN support will be disabled.
Setting up Install Process
Parsing package install arguments
#以下为软件依赖信息详情
Resolving Dependencies
--> Running transaction check
---> Package httpd.i386 0:2.2.3-22.el5 set to be updated
--> Processing Dependency: libapr-1.so.0 for package: httpd
--> Processing Dependency: libaprutil-1.so.0 for package: httpd
--> Running transaction check
---> Package apr.i386 0:1.2.7-11 set to be updated
---> Package apr-util.i386 0:1.2.7-7.el5 set to be updated
--> Processing Dependency: libpq.so.4 for package: apr-util
--> Running transaction check
---> Package postgresql-libs.i386 0:8.1.11-1.el5_1.1 set to be updated
```

```
--> Finished Dependency Resolution

Dependencies Resolved
#以下为要更新的软件包详情
#其中包括软件包名称、硬件平台、版本、使用的源名称、软件包大小等信息
=======================================================================
 Package              Arch       Version               Repository   Size
=======================================================================
Installing:
 httpd                i386       2.2.3-22.el5              cdrom       1.2 M
#以下为需要更新的依赖软件包详情（内容与软件包详情相同）
Installing for dependencies:
 apr                  i386       1.2.7-11                  cdrom       123 k
 apr-util             i386       1.2.7-7.el5               cdrom        76 k
 postgresql-libs      i386       8.1.11-1.el5 1.1          cdrom       196 k

#以下为需要安装、更新、删除的软件包统计信息
Transaction Summary
=======================================================================
Install         4 Package(s)
Update          0 Package(s)
Remove          0 Package(s)

Total download size: 1.6 M
#命令提示用户阅读以上信息，并要求用户确认，本例中输入 y 并按下 Enter 键
Is this ok [y/N]: y
#同意 yum 的更新操作后，yum 会立即下载需要安装的软件包
Downloading Packages:
-----------------------------------------------------------------------
---------
Total                                     342 MB/s | 1.6 MB    00:00
#下载完成后，立即执行测试
Running rpm check debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
#通过测试之后立即开始安装过程
Running Transaction
  Installing    : apr                                          [1/4]
  Installing    : postgresql-libs                              [2/4]
  Installing    : apr-util                                     [3/4]
  Installing    : httpd                                        [4/4]

#安装完成后，输出相关提示信息
Installed: httpd.i386 0:2.2.3-22.el5
Dependency Installed: apr.i386 0:1.2.7-11 apr-util.i386 0:1.2.7-7.el5
postgresql-libs.i386 0:8.1.11-1.el5_1.1
Complete!
```

在上面的示例命令中，yum 会先查找可用的 yum 源，并使用源计算软件包的依赖关系。然后提示用户，得到用户的允许后，yum 将从源下载软件包并执行测试。测试通过后，yum 将立即开始安装软件包，最后输出提示信息。

（2）如果确信使用的 yum 源可靠，可以使用选项 y 授权 yum 自动操作：

```
#使用选项 y 授权 yum 自动安装软件
# yum -y install httpd
```

使用以上命令安装软件包时，yum 将不会提示用户，计算依赖后，直接下载安装软件包。

9.7.5　利用 yum 卸载软件包

与安装软件包一样，yum 卸载软件包也非常简单，直接使用 remove 参数即可。本小节将简单介绍如何利用 yum 卸载系统中的软件包。

【命令格式】

```
yum [-y] remove soft_package_name
```

与安装软件包类似，选项 y 将授权 yum 自动操作。

【用法示例】

（1）例如使用 yum 删除软件包：

```
#使用 yum 删除软件包 httpd
# yum remove httpd
Loaded plugins: rhnplugin, security
This system is not registered with RHN.
RHN support will be disabled.
Setting up Remove Process
Resolving Dependencies
--> Running transaction check
---> Package httpd.i386 0:2.2.3-22.el5 set to be erased
--> Finished Dependency Resolution
……
```

卸载软件与安装软件的过程非常类似，yum 先检查软件包的依赖性，然后询问用户是否需要卸载列表中的软件包，最后将软件包从系统中卸载。

（2）如果确信需要执行的所有操作，可以使用选项 y 授权 yum 自动操作：

```
#使用选项 y 授权 yum 自动卸载软件
# yum -y remove httpd
……
```

注意：不建议使用 yum 工具自动卸载与系统相关的软件包，众多的软件依赖关系可能会导致系统的某些功能无法使用。

9.7.6　安装、卸载软件包组

软件包组是发行版自定义的软件包集合，软件包组会显示在操作系统安装过程中的软

件定制界面中。本小节将简单介绍如何利用 yum 安装、卸载软件包组。

【命令格式】

```
yum [grouplist|groupinstall|groupremove] group_name
```

软件包组的操作中，参数 grouplist 表示列出所有的软件包组，groupinstall 表示安装指定的软件包组，groupremove 表示卸载指定的软件包组。

【用法示例】

（1）如果不知道源提供的软件包组名称，可以通过命令 yum grouplist 查看 yum 源提供的软件包组：

```
#使用 yum 命令查看源提供的软件包组
# yum grouplist
#查询可用的源
Loaded plugins: rhnplugin, security
This system is not registered with RHN.
RHN support will be disabled.
Setting up Group Process
#以下为软件包组名称
#以下为系统中已安装的软件包组名称
Installed Groups:
   Administration Tools
   Editors
   GNOME Desktop Environment
   Games and Entertainment
……
#以下为源提供但系统中未安装的软件包组
Available Groups:
   Authoring and Publishing
   DNS Name Server
   Development Libraries
   Development Tools
   Engineering and Scientific
……
Done
```

从上面的示例命令输出中可以看出，这个列表与安装操作系统时的软件定制列表相同（与字符安装界面中的软件定制列表完全一样），可以通过 yum 管理这些软件包组。

（2）安装软件包组时，只需要使用 groupinstall 参数加软件包组名称即可：

```
#使用 yum 安装软件包组 DNS Name Server
#注意软件包组名称需要放在双引号中
# yum groupinstall "DNS Name Server"
Loaded plugins: rhnplugin, security
This system is not registered with RHN.
RHN support will be disabled.
Setting up Group Process
Resolving Dependencies
```

```
--> Running transaction check
---> Package bind-chroot.i386 30:9.3.4-10.P1.el5 set to be updated
---> Package bind.i386 30:9.3.4-10.P1.el5 set to be updated
--> Finished Dependency Resolution
……
```

安装过程与安装单个软件包类似，先检查依赖性关系，然后询问用户是否进行安装，最后安装软件包。

（3）卸载功能性软件包组时，可以使用 groupremove 参数。例如要删除 DNS Name Server：

```
#使用 yum 命令删除软件包组 DNS Name Server
# yum groupremove "DNS Name Server"
Loaded plugins: rhnplugin, security
This system is not registered with RHN.
RHN support will be disabled.
Setting up Group Process
Resolving Dependencies
--> Running transaction check
---> Package bind-chroot.i386 30:9.3.4-10.P1.el5 set to be erased
---> Package bind.i386 30:9.3.4-10.P1.el5 set to be erased
--> Finished Dependency Resolution
……
```

软件卸载过程与单个软件包卸载过程相似，不同的是此处使用的是由软件自己提供的软件包名称。

使用 yum 工具卸载软件包，与使用 rpm 命令卸载软件包一样，仅能卸载指定名称的软件包，并不能卸载其所依赖的软件包。

提示：yum 命令还有许多用法，感兴趣的读者可以阅读相关文档了解。

9.8　小　　结

- 9.1 节介绍了数据备份的相关内容，包括数据的重要性、存放备份数据的介质，以及制定一个备份策略时需要注意的事项等。
- 9.2 节讲解了如何使用系统自带的 tar 工具归档数据。tar 是 Linux 系统中最为常用的备份工具，目前互联网上的许多文档、软件都使用此工具归档和压缩，因此学会 tar 的用法对文件和目录的归档非常重要。
- 9.3 节介绍了 Linux 系统中的另一个重要的归档文件命令 cpio。cpio 命令虽然古老，但有时仍然需要使用这个命令，因此初学者应该了解这个命令。
- 9.4 节简单介绍了 Linux 中的压缩命令和整盘备份工具 dd。包括使用 gzip 和 bzip2 命令压缩归档的文件，使用 dd 命令对磁盘进行整盘备份，整盘复制，制作光盘镜像，产生一个指定大小的文件等。dd 命令是 Linux 系统中重要的数据备份命令，

初学者应该掌握这个命令的用法。

- 9.5 节介绍了 Linux 系统中的 RPM 包管理器。RPM 软件包管理器是最早的软件管理软件之一，现在已经广泛应用于各大发行版中。初学者应该掌握使用 RPM 包管理器管理软件的方法。
- 9.6 节主要介绍了如何在 Linux 系统中编译安装软件。可能许多用户不会用到此部分内容，读者按需要学习即可。
- 9.7 节讲解了 Linux 系统中的 yum 软件包管理器。yum 是一个基于 RPM 包管理器的自动化管理工具，初学者应该掌握 yum 工具的使用方法。

本章主要介绍了 Linux 系统中的数据备份、应用程序管理相关命令和工具，对于大多数学习者而言，本章的内容十分有用，读者要掌握本章中的大部分内容。

第 10 章　网 络 管 理

与前些年相比，现在越来越多的企业都在使用 Linux 系统，甚至将其作为自己主要服务器的操作系统。这得益于 Linux 系统中有许多可以免费获得和使用的服务软件，例如域名服务、FTP、HTTP 等，这些网络服务都依赖于 Linux 系统的网络功能，因此网络管理十分重要。无论初学者出于什么目的学习 Linux，网络管理都是必学的内容之一。

本章将介绍 Linux 系统的网络设置、网络冗余、网络工具和网络故障排除的基本思路，涉及的主要内容如下。

- 查看网络的基本信息，包括网络接口、路由表及主机名称等。
- 使用命令配置 IP 地址、子网掩码、默认网关、主机名、与网络设置有关的系统配置文件等。
- 使用 bonding 配置网络冗余、网络负载均衡。
- 介绍 Linux 系统中的网络工具，以及如何使用这些工具排除网络故障。

提示：在学习本章之前读者可能需要了解计算机网络方面的一些基础知识。按学习目的不同，可能需要了解基本理论知识，包括 OSI 七层网络模型、IP 四层网络模型及各层的寻址方式、以太网、IPv6 等。还需要了解一些网络应用基础，按学习目的不同，可能包括以太网交换机的工作原理、路由器的工作原理、VLAN（Virtual Local Area Network，虚拟局域网）、VLSM（Variable Length Subnet Mask，可变长子网掩码）、以太通道（Ethernet Channel，CISCO 的聚合链路）等实用技术。

10.1　网络接口配置命令

网络接口的重要性不言而喻，如果计算机要与外部通信，就必须使用网络接口。网络接口（通常称为网卡，也称网络接口卡）是目前计算机中最主要的网络连接设备。默认情况下，一个网络接口卡对应一个网络连接。如果需要将计算机与网络相联，除非使用了 DHCP 服务器（DHCP 服务器能自动为网络中的计算机分配 IP 地址、子网掩码等信息），否则需要手动配置网络接口。本节将简单介绍 Linux 系统中常用的网络接口配置命令。

10.1.1　查看网络接口信息

在 RHEL5.3 中，使用 ethX 的方式命名网络接口，其中 X 表示网络接口编号（有些系

统并不是以此方法命名网络接口，读者可阅读相关说明文件）。如果要正确地连接网络，还需要为相应的网络接口配置 IP 地址等信息。在学习如何配置网络接口之前，应该了解如何查看网络接口信息，以便于确认可用的网络接口。本小节将简单介绍如何查看网络接口信息。

1．ifconfig命令

查看系统中所有可用的网络接口信息，可以使用命令 ifconfig。命令中的 if 是 interface 的缩写，其中文含义是接口，读者需要区分该命令与 Windows 系统中的 ipconfig 命令（该命令用于查看网络连接的 IP 地址等信息）。

【命令格式】

ifconfig 命令既可以查看网络接口信息，也可以配置网络接口。查看网络接口信息时其格式如下：

```
ifconfig [option] [interface]
```

使用 ifconfig 命令查看网络接口信息时，可以使用 interface 参数指定要查看的网络接口，也可以不使用此参数查看所有网络接口。

【常用选项】

查看网络接口信息时，ifconfig 命令的常用选项只有一个 a，其功能是显示系统中的所有接口，包括被禁用的接口。

【用法示例】

（1）如果不使用任何选项和参数，ifconfig 命令将显示当前系统中的所有活动接口。例如：

```
#使用 ifconfig 命令查看系统中的所有活动接口
# ifconfig
#以下为接口信息
eth0     Link encap:Ethernet  HWaddr 00:0C:29:28:68:0A
         inet addr:192.168.118.234  Bcast:192.168.118.255  Mask:255.255.255.0
         inet6 addr: fe80::20c:29ff:fe28:680a/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:554302 errors:0 dropped:0 overruns:0 frame:0
         TX packets:19112 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:43362431 (41.3 MiB)  TX bytes:4886841 (4.6 MiB)
         Interrupt:67 Base address:0x2000

lo        Link encap:Local Loopback
         inet addr:127.0.0.1  Mask:255.0.0.0
         inet6 addr: ::1/128 Scope:Host
         UP LOOPBACK RUNNING  MTU:16436  Metric:1
         RX packets:149 errors:0 dropped:0 overruns:0 frame:0
         TX packets:149 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:0
         RX bytes:21224 (20.7 KiB)  TX bytes:21224 (20.7 KiB)
```

在上面的示例命令输出中可以看到，系统中有两个活动的网络接口 eth0 和 lo，除此之

外，还有这两个网络接口的详细信息。

示例命令输出的接口信息非常多，表示的含义如表 10.1 所示。

表 10.1　ifconfig命令输出选项的含义

选　　项	选 项 含 义
Link encap	连接类型。通常为 Ethernet，即以太网接口
HWaddr	网络接口使用的 MAC 地址，这个地址通常被固化在网卡的芯片内
inet addr	当前接口使用的 IPv4 地址
Bcast	接口使用的广播地址
Mask	IPv4 地址对应的子网掩码
inet6 addr	IPv6 地址及掩码
UP	表示当前接口已经连接
RUNNING	表示网线已经连接到网络接口
MULTICAST	当前接口支持组播
BROADCAST	当前接口支持广播
MTU	最大传输单元（即数据包最大字节数），这个值的大小通常与网络类型相关
RX packets	接口启用至今发送的数据包总数
TX packets	接口启用至今接收到的数据包总数
errors	接口启用至今错误包总数
dropped	接口启用至今丢弃包总数
overruns	接口溢出包总数
collisions	表示信号冲突的次数，通常与物理链路使用情况有关
txqueuelen	表示网络接口使用的缓冲区大小
RX bytes	接口发送的总字节数
TX bytes	接口接收到总字节数
Interrupt	表示该设备的 IRQ 中断值

注意： 网络接口信息中的错误和丢弃数据包数量，通常与物理链路质量相关。溢出数据包总数通常与缓冲区大小及收发数据包的速度有关。

上面的命令输出中，还显示了一个特殊的接口 lo，这是一个环回接口（Loopback）。顾名思义，这个接口的数据总是指向自己，即发送到这个接口的所有数据包都会发送到系统本身。许多服务都依赖环回接口 lo，例如 sendmail 等。

（2）如果系统中有网络接口被禁用，使用命令 ifconfig 将不会显示这些网络接口信息，此时可以配合使用选项 a 显示所有的网络接口信息。

显示系统中所有的网络接口的代码如下：

```
#使用选项 a 显示系统中的所有网络接口
# ifconfig  -a
eth0      Link encap:Ethernet  HWaddr 00:0C:29:28:68:0A
          inet addr:192.168.118.234  Bcast:192.168.118.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe28:680a/64 Scope:Link
……
#eth1 为被禁用的接口
eth1      Link encap:Ethernet  HWaddr 00:0C:29:28:68:14
```

```
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:33 errors:0 dropped:0 overruns:0 frame:0
          TX packets:217 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3707 (3.6 KiB)  TX bytes:39785 (38.8 KiB)
          Interrupt:67 Base address:0x2080

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
……
#sit0 用于在 IPv4 和 IPv6 网络间传输数据
sit0      Link encap:IPv6-in-IPv4
          NOARP  MTU:1480  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

上面的示例命令输出中，除了接口 eth0 和环回接口 lo 之外，还有一个没有启用的以太网接口 eth1 和一个名为 sit0 的接口。网络接口 eth1 由于被禁用，因此没有显示出 IP 地址等信息。

网络接口 sit0 是一个非常特殊的接口，它就像在 IPv4 和 IPv6 网络之间传输数据包的隧道（即将 IPv6 数据包转换后传入 IPv4 网络）。虽然在许多系统中都存在这个接口，但通常这个接口都没有使用。

（3）除此之外，还可以在使用 ifconfig 命令时，指定要查看的网络接口。例如查看网络接口 eth0，使用命令 ifconfig eth0。

其输出结果与上个示例中的 eth0 相同，此处不再赘述。

2．使用ip命令查看网络接口的IP地址

有时可能只想看到网络接口使用的 IP 地址，这时可以使用 ip 命令查看网络接口的相关信息。

例如要查看当前系统中网络接口的 IP 地址信息：

```
#使用 ip address show 命令查看网络接口的 IP 地址信息
# ip address show
1: lo: <LOOPBACK,UP,LOWER UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid lft forever preferred lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER UP> mtu 1500 qdisc pfifo fast qlen
1000
    link/ether 00:0c:29:28:68:0a brd ff:ff:ff:ff:ff:ff
    inet 192.168.118.234/24 brd 192.168.118.255 scope global eth0
    inet6 fe80::20c:29ff:fe28:680a/64 scope link
       valid lft forever preferred lft forever
3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo fast qlen 1000
    link/ether 00:0c:29:28:68:14 brd ff:ff:ff:ff:ff:ff
```

```
4: sit0: <NOARP> mtu 1480 qdisc noop
   link/sit 0.0.0.0 brd 0.0.0.0
```

在上面的示例命令中，ip address show 以非常简捷的方式输出了当前系统中所有的网络接口信息。

读者可能已经注意到接口后面的 qdisc pfifo_fast 等内容，这是接口使用的队列相关信息。使用队列可以控制接口的流量，流量控制不是本书讨论的内容，读者可以自行阅读相关文档和说明。

【如何表示一个网络】

许多书籍中都使用一个 IP 地址表示网络，例如网络 192.168.1.0 表示 IP 地址为 192.168.1.1 至 192.168.1.255 的 C 类网络。对这个 C 类网络可以有好几种称谓，常见的表述方法有：192.168.1.0 网段、网络号为 192.168.1.0、192.168.1.0 子网等。

如果不是标准网络，就需要使用 IP 地址和子网掩码按位与运算得到网络号（相当于按位乘法运算）。例如计算 IP 地址为 172.17.1.55（IP 地址为 B 类），子网掩码为 255.255.255.0（但使用 C 类的子网掩码）的网络号。先将 IP 地址和子网掩码的第 1 段转换成二进制数。172 对应的二进制数为 10101100，255 对应的二进制数为 11111111，按位与运算后的结果为 10101100（对应的十进制数为 172）。依此类推，将剩下 3 段按位与，得出的结果为 172.17.1.0，这就是 IP 地址对应的网络号。

说明： 由于目前绝大多数企业和用户都使用以太网（以太网是局域网技术之一，其他的局域网技术还有令牌环、分布式光纤接口和 ATM），因此本书仅以以太网为基础讲解网络管理，对于其他网络类型，读者可以阅读相关文档。

10.1.2　配置网络接口

除非使用了 DHCP，否则只是网线与网卡连接是无法连接到网络的，还需要修改网络接口的配置，为其添加 IP 地址、子网掩码等，才能连接到网络。本小节将简单介绍如何配置网络接口。

1. 使用ifconfig命令配置网络接口

当系统中存在多个网络接口时，如果未正确配置 IP 地址等信息，通常是无法判断哪个网络接口连接到了哪个网络上的。另一种情况是向系统添加了新的网卡，系统启动后，网络接口的编号发生了变化（除非原来系统中的网卡配置文件中保存了网卡的 MAC 地址），无法判断哪个网络接口对应哪个网络。

如果遇到了上面讲述的这两种情况，建议先使用 ifconfig 命令配置网络接口，并判断网络接口对应的网络，然后将配置写入配置文件。

【命令格式】

使用 ifconfig 命令配置网络接口时，其格式如下：

```
ifconfig <intelface> <address> netmask <netmask> [up|down]
ifconfig <intelface> <address> </prefixlen> [up|down]
```

使用 ifconfig 命令只能临时修改网络接口的设置，系统重新启动后，这些设置将会丢失。

【用法示例】

（1）如果要设置的网络接口已经处于活动状态，可以直接使用 ifconfig 命令为其指定 IP 地址、子网掩码。

例如要设置网卡 eth1 的 IP 地址、子网掩码：

```
#使用 ifconfig 命令设置 eth1 的 IP 地址和子网掩码
#IP 地址为 192.168.204.200，子网掩码为 255.255.255.0
# ifconfig eth1 192.168.204.200 netmask 255.255.255.0
```

（2）如果觉得上面的命令输入起来非常麻烦，可以使用比较简捷的方法：

```
#使用长度表示子网掩码
# ifconfig eth1 192.168.204.200/24
```

（3）如果网络接口处于非活动状态，可以使用 up 参数让配置立即生效，并使用新的配置信息启动网络连接：

```
#使用参数 up，配置并立即启动网络接口
# ifconfig eth1 192.168.204.200 netmask 255.255.255.0 up
```

当然上面的命令也可以写为如下形式：

```
#使用长度方式表示子网掩码
# ifconfig eth1 192.168.204.200/24 up
```

使用 ifconfig 命令设置网络接口后，接下来就可以使用 ping 命令验证网络接口 eth1 对应的网络是否为 192.168.204.0 了。

小知识： 以长度方式表示子网掩码时，需要将子网掩码中的所有数字都转换成二进制数，使用二进制数 1 的位数表示子网掩码。由于十进制数 255 转换成二进制数为 8 个 1，因此 24 位表示的子网掩码应该是 255.255.255.0。

2．保存设置到网络接口配置文件

网络接口的配置文件位于目录/etc/sysconfig/network-scripts/中，按网络接口的名称不同，文件名称为 ifcfg-ethX，其中 X 为网络接口编号。除此以外，还有一个名为 ifcfg-lo 的文件，该文件是环回接口的配置文件。

查看网络接口 eth1 的配置文件如下：

```
#使用 cat 命令查看网络接口 eth1 的配置文件
# cat /etc/sysconfig/network-scripts/ifcfg-eth1
# Advanced Micro Devices [AMD] 79c970 [PCnet32 LANCE]
DEVICE=eth1
BOOTPROTO=none
ONBOOT=yes
HWADDR=00:0c:29:28:68:0a
NETMASK=255.255.255.0
IPADDR=192.168.204.200
```

```
GATEWAY=192.168.204.1
TYPE=Ethernet
```

这是一个典型的使用静态 IP 地址的网络接口配置文件内容。在这个配置文件中以“#”开头的为注释行，注释行仅仅提供注释说明的功能，系统在读取配置文件时会忽略注释行。

网络接口配置文件中的有效行都有一个固定的格式，即“配置项=配置项的值”。常见的配置项的具体含义和可选值如下。

- DEVICE：网络接口的名称，通常由系统自动生成，无须手动配置。此处的网络接口名称应该与配置文件名相对应。
- BOOTPROTO：表示此网络接口的配置方式，即开启该网络接口之后应该如何获取其设置。
- ONBOOT：系统启动时是否需要启动该连接。yes 表示启动，no 表示关闭。
- HWADDR：这是一个可选配置项，表示网络接口使用的 MAC 地址。某些系统会自动生成此配置项及其值。通常不需要修改该配置项的值。
- NETMASK：网络接口使用的子网掩码。
- IPADDR：网络接口使用的 IP 地址。
- GATEWAY：网络接口对应网络的默认网关。
- TYPE：该连接的类型，最常见的值为 Ethernet，表示这是一个以太网接口。

上面的配置项中，HWADDR 和 TYPE 都是由系统自动生成的，通常这两个选项都可以不必设置。

在上面列举的配置项中，有一个比较特殊的配置项 BOOTPROTO，这个配置项可以使用的值及其含义如下。

- dhcp：这个值表示该连接启动时应该尝试从 DHCP 服务器处获取此连接的初始配置信息。从 DHCP 服务器中获取的配置信息包括 IP 地址、子网掩码、默认网关及 DNS 服务器地址（DNS 服务器用于解析域名和 IP 地址间的对应关系）等。
- static：表示该连接应该使用配置文件中的设置。使用此值时，需要在配置文件中设置静态的 IP 地址、默认网关（可选）、子网掩码。
- none：表示不使用 DHCP 获取连接初始化信息。该值可以用于接口使用静态 IP 地址时，也可以表示网络接口有其他用途，不需要使用 IP 地址等设置时（例如网络接口处于 bonding 中时）。

注意：在使用配置文件设置连接初始化信息时，应该特别注意的是配置项名称应该都使用大写字母，而配置项的值都应该使用小写字母。如果大小写错误，网络接口启动时会引发一些意外的错误。

了解配置文件中各配置项的含义后，读者可以按上面的示例，自行修改配置文件中的设置，此处不再赘述。

如果需要使用 DHCP 获取网络接口的配置信息，配置文件的内容如下：

```
#使用 cat 命令查看网络接口 eth1 使用 DHCP 获取设置时的配置内容
# cat /etc/sysconfig/network-scripts/ifcfg-eth1
# Advanced Micro Devices [AMD] 79c970 [PCnet32 LANCE]
DEVICE=eth0
```

```
#HWADDR 配置项仍然是可选项
HWADDR=00:0c:29:28:68:14
BOOTPROTO=dhcp
TYPE=Ethernet
```

读者可以按自己的实际情况，参考此处的两个示例设置网络接口的配置文件。

3. 使用工具设置网络接口

除了使用命令和工具设置网络接口以外，还可以使用某些发行版自带的工具设置网络接口。例如 Red Hat Enterprise Linux 4 中自带的 netconfig、SUSE Enterprise Linux 中自带的 yast2 等。

在 RHEL5.3 中，网络接口配置工具都集成到 setup 工具集中（setup 工具集的主要作用是快速设置操作系统）。此处以使用 setup 工具设置网络接口为示例，讲解如何使用命令提示符中的工具。

运行命令 setup 就可以启动 setup 工具集界面。setup 工具集界面如图 10.1 所示。



图 10.1　setup 工具集界面

此时使用上下方向键选中 Network configuration（网络设置），然后使用 Tab 键选中 Run Tool（运行工具）并按下 Enter 键就可以启动网络配置工具。

启动网络配置工具之前会询问要修改的配置，有两个选项：Edit a device params（此项用于设置网络接口）和 Edit DNS configuration（此项用于设置 DNS 和主机名）。

此时使用上下方向键选中 Edit a device parsms，并按下 Enter 键将会出现网络接口设备选择界面，如图 10.2 所示。

选中相应的网络接口，按下 Enter 键，即可出现相应的网络接口配置界面，如图 10.3 所示。

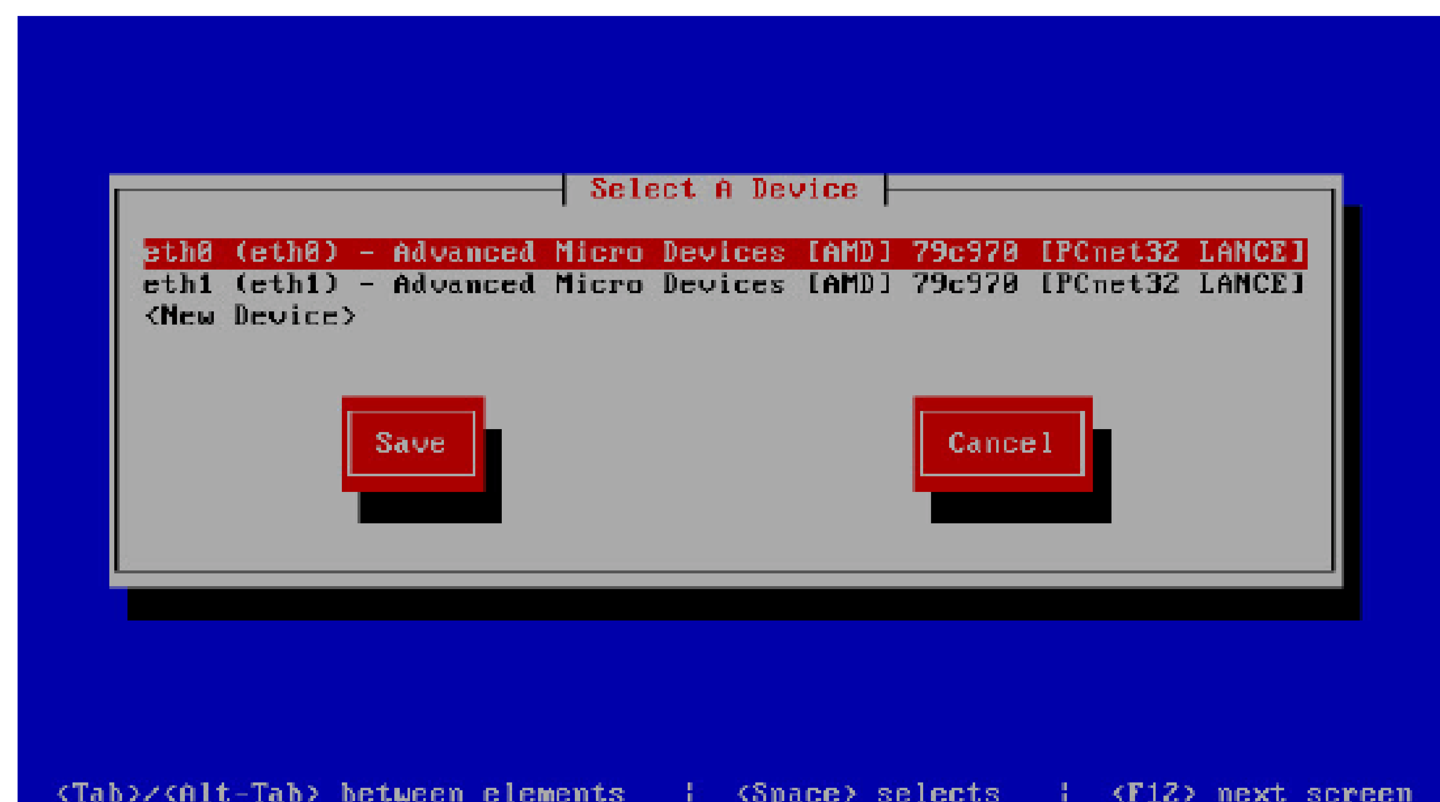


图 10.2　网络接口设备选择界面

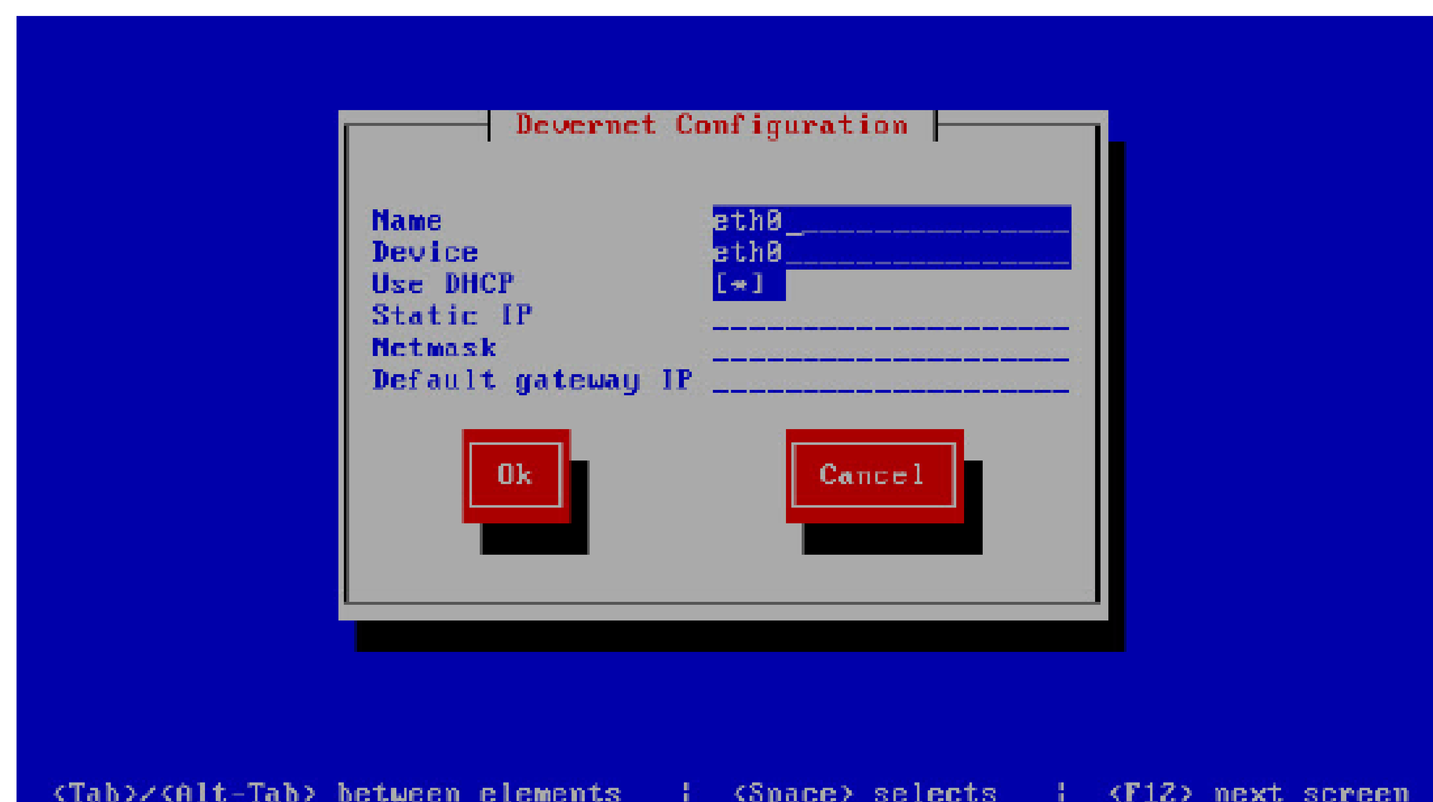


图 10.3　网络接口配置界面

此时可以使用 Tab 键切换要选择的配置项，使用空格键选中或取消单选框。在图 10.3 所示的配置项中，Name 和 Device 通常由系统自动生成，不用配置，其他选项都需要按实际情况设置。

如果该网络接口要使用静态 IP，可以使用 Tab 键，将光标移动到 Use DHCP，并按下空格键。然后在 Static IP、Netmask 和 Default gateway IP 中输入该接口使用的 IP 地址、子网掩码和默认网关。设置完成之后，按 Tab 键选择 OK 按钮，保存并退出工具即可。

使用 setup 工具集配置网络接口的 IP 地址等信息后，setup 会自动将设置保存到配置文件中，可以使用查看网络接口配置文件的方法验证。

10.1.3　重新启用网络接口

虽然已经通过修改配置文件、set 工具集等方法保存了网络接口的设置，但这些设置并不会立即生效。这时需要手动重新启用相关网络接口，让系统读取新的配置文件，之后才能生效。要重新启用网络接口，可以使用两种方法实现：重新启动系统网络服务、重新启动指定的网络接口。

1．重新启动网络服务

在 RHEL5.3 中，有一个名为 network 的系统服务。这个服务的主要作用是初始化系统网络，为系统提供网络支持。系统启动时，network 服务会检查系统中网络接口的情况，并启用这些连接。可以利用重新启动这个服务的方法重新启用网络接口。

（1）重新启动网络服务：

```
#使用关闭、启动网络服务的方法重新启用网络接口
# service network stop
Shutting down interface eth0:                          [  OK  ]
Shutting down interface eth1:                          [  OK  ]
Shutting down loopback interface:                      [  OK  ]
# service network start
Bringing up loopback interface:                        [  OK  ]
Bringing up interface eth0:                            [  OK  ]
Bringing up interface eth1:
Determining IP information for eth1... done.
                                                       [  OK  ]
```

从上面的命令输出中可以看出，服务关闭时会停止所有的网络接口，服务启动时又会开启所有的网络接口。在这个过程中，系统会重新读取并使用网络接口的配置文件设置网络接口。

（2）如果系统不支持命令 service，可以使用以下命令代替：

```
#使用绝对路径的方式重新启动网络服务
# /etc/init.d/network restart
Shutting down interface eth0:                          [  OK  ]
Shutting down interface eth1:                          [  OK  ]
Shutting down loopback interface:                      [  OK  ]
Bringing up loopback interface:                        [  OK  ]
Bringing up interface eth0:                            [  OK  ]
```

```
Bringing up interface eth1:
Determining IP information for eth1... done.
                                                      [  OK  ]
```

2．重新启动网络接口

要让系统重新读取已经设置好的网络接口配置文件，也可以使用重新启动网络接口的方法。

（1）重新启用网络接口，也可以使用 ifconfig 命令。例如要重新启用网络接口 eth1：

```
#使用 ifconfig 命令的 down 和 up 参数重新启用网络接口 eth1
# ifconfig eth1 down
# ifconfig eth1 up
```

执行以上命令后，网络接口 eth1 就已经重新启用了。

（2）除 ifconfig 命令以外，还可以使用命令 ifdown 和 ifup。这两个命令的功能分别是关闭和开启参数指定的网络接口。

例如要重新启动网络接口 eth1：

```
#使用 ifdown 和 ifup 命令重新启用网络接口
# ifdown eth1
# ifup eth1
```

10.1.4　配置 DNS 服务器地址

细心的读者可能已经发现还没有设置 DNS 服务器地址（DNS，Domain Name System），这是每一个使用 Internet 的计算机都需要配置的内容。计算机只有通过 DNS 才能查找到域名对应的 IP 地址，因此在计算机网络中十分重要。

提示：Linux 中也提供了 DNS 相关的命令，但通常只用这些命令排除网络故障（例如测试域名服务器能否正常解析域名等）。关于这些命令的使用将在 10.4 节中介绍。

与网络接口的配置相同，也需要将 DNS 服务器地址保存在文件中，并且也可以使用修改文件和使用工具配置两种方法。本小节将简单介绍如何配置 DNS 服务器地址。

1．域名服务器配置文件

在 Linux 系统中，域名服务器的配置文件是/etc/resolv.conf。默认情况下，域名配置文件中只保留了一个搜索路径。

（1）查看这个文件的内容：

```
#使用 cat 命令查看域名服务器的配置文件
# cat /etc/resolv.conf
#下面这个配置行表示在当前主机使用的域 localdomain 中搜索
search localdomain
```

（2）如果要添加 DNS 服务器地址，需要使用以下格式：

```
#添加 DNS 服务器地址的格式
```

```
#其中 IPaddress 为 DNS 服务器的 IP 地址
nameserver IPaddress
```

此处以两个私有域名服务器 IP 地址为例，修改文件 resolv.conf 的内容如下：

```
#查看修改后的域名服务器配置文件的内容
# cat /etc/resolv.conf
#使用“#”注释文件原有内容
#search localdomain
#添加两个私有域名服务器：192.168.111.200 和 192.168.121.200
nameserver 192.168.111.200
nameserver 192.168.121.200
```

读者可以按照以上格式添加多个 DNS 服务器的 IP 地址。

（3）Linux 系统并没有对添加的 DNS 服务器 IP 地址的个数作限制，但当设置的 DNS 服务 IP 地址超过 3 条时，通常只有前 3 条起作用。

提示： 域名服务器的 IP 地址在全国各个省市均不同，通常可以向当地电信运营商或互联网提供商询问，对于拥有大型网络的企业而言，也可以自行架设。

2．使用工具配置DNS服务器地址

除了修改域名服务配置文件的方法以外，也可以利用 setup 等工具设置主机的 DNS 服务器 IP。本例中将以 RHEL5.3 自带的 setup 工具为例，介绍如何使用工具配置 DNS 服务器地址。

（1）运行 setup 命令，系统将弹出 setup 工具。在工具主界面中，使用上下方向键选中 Network configuration 并使用 Tab 键选择 Run Tool，然后按下 Enter 键。在弹出的选择页面中选中 Edit DNS configuration（编辑 DNS 设置）选项后，按 Enter 键即可进入 DNS 配置界面，如图 10.4 所示。

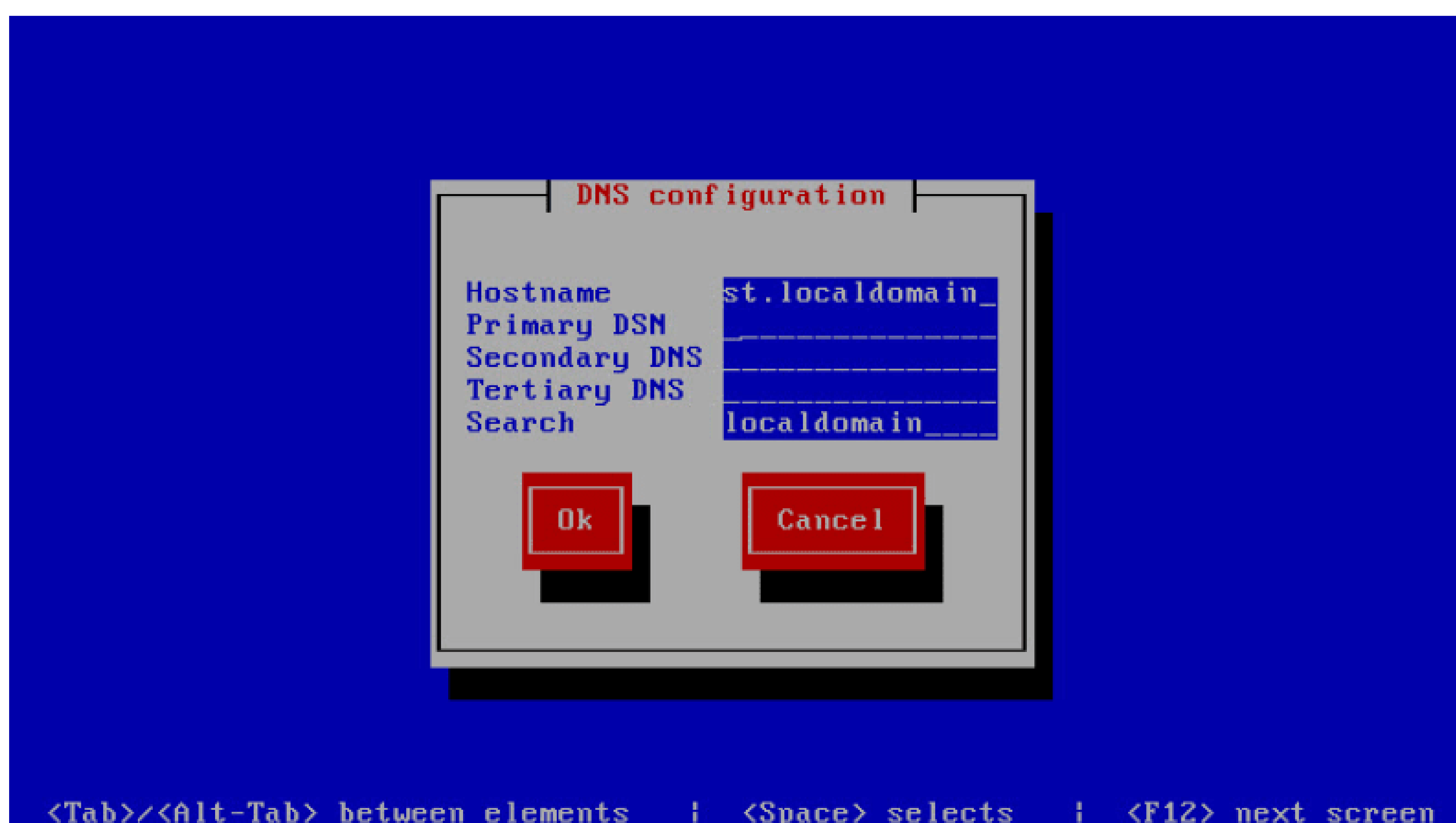


图 10.4　DNS 配置界面

（2）在 DNS 配置界面中的 Primary DNS（主要 DNS 服务器），Secondary DNS（次要 DNS 服务）和 Tertiary DNS（第三 DNS 服务）后面添加相应的 DNS 服务器 IP 地址即可。

也可以在 Search 后面添加搜索路径，完成后选择 OK 按钮，保存并退出即可生效。

（3）在 DNS 配置界面中，Hostname 设置项表示设置主机名称。如果需要设置此项，请参照 10.3 节中的相关内容。

注意： 无论使用何种方式设置 DNS 服务器 IP 地址，都会立即生效，并不需要系统重新读入这些设置。

10.2　路由命令 route

路由是网络层中最重要的概念，这是因为任何主机如果需要与外部网络通信，就必须使用路由（大部分计算机都使用默认路由）。简单地理解，路由就是计算机或路由器计算数据包路径的依据。借助于路由，计算机或路由器能计算目标主机的位置，并通过路由功能将数据包安全送达。

小知识： 路由器是为数据包寻找最佳传输路径的网络设备，其工作原理与计算机相同。

为了说明路由在 Linux 系统中的应用，此处引入目前大多数企业正在使用的安全模型，其结构如图 10.5 所示。

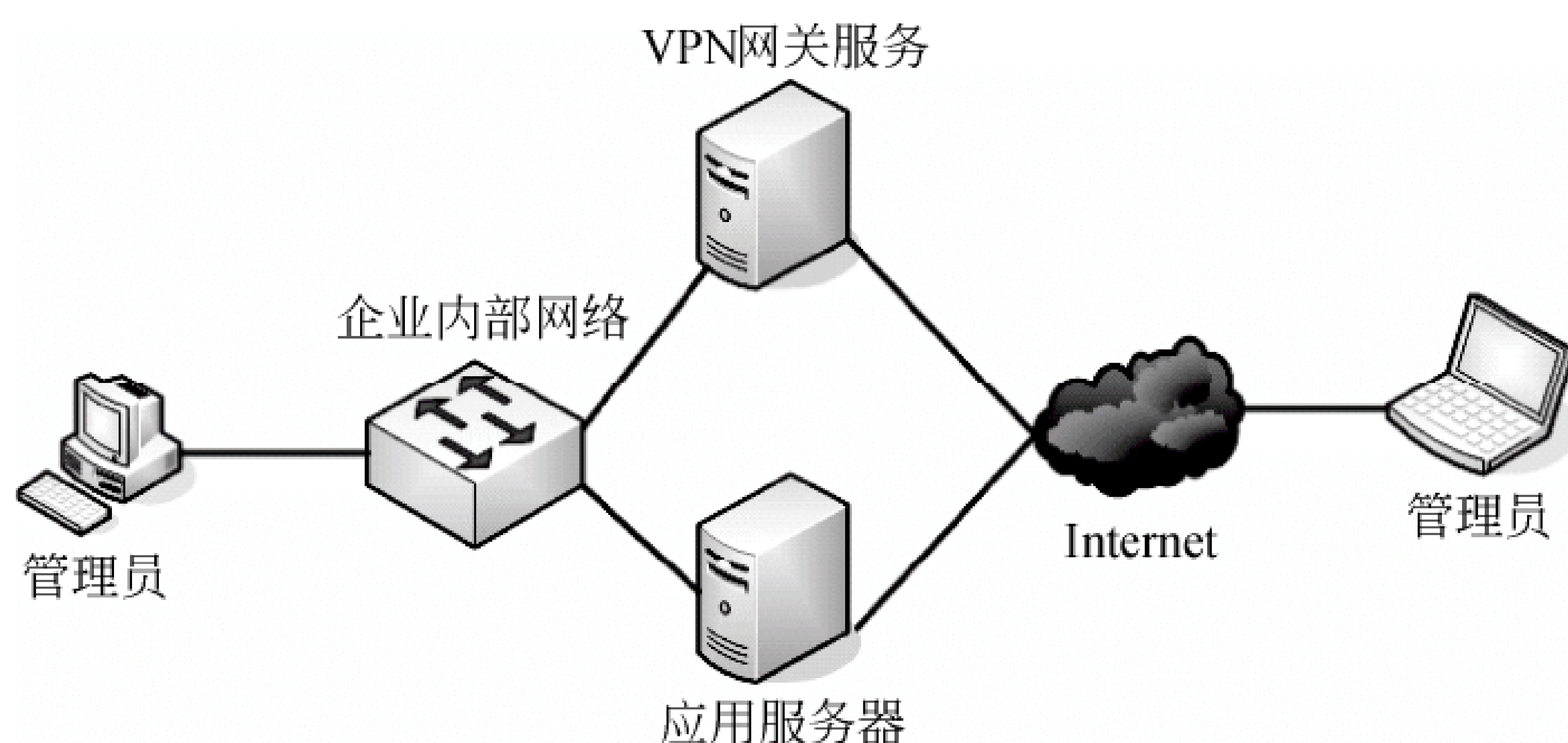


图 10.5　简单安全模型

图 10.5 是一个被广泛应用的安全模型。在这个安全模型中，应用服务器使用两个不同的网络接口分别与企业内部网络和 Internet 相连。对于这种情况，管理数据包从哪个网络接口发送出去，又从哪个路径到达目标主机非常重要。如果管理数据包从不安全的路径到达目标主机，非常危险。

例如管理数据包从与 Internet 相连的网络接口进入时，无法确认管理数据包在传输过程中是否被篡改过，也无法确认管理数据包是否来自一个欺骗者。这时管理员需要控制数据包经过的路径（即路由管理），让所有管理数据包都通过安全的路径到达服务器。

在图 10.5 所示的模型中，管理员需要让所有管理数据包都通过企业内部网络。具体做法是管理员登录应用服务器，为管理员所在网络和登录 VPN 后使用的网络添加路由。按管理方式不同，可能还需修改 SSH 服务的配置文件、设置防火墙控制管理数据包，这些不是本书讨论的重点，感兴趣的读者可以参考相关书籍。

🔔小知识：VPN（Virtual Private Network，虚拟专用网络或虚拟专网）是一种通过公共网络建立安全连接隧道的技术，所有穿过隧道的数据包都需要加密，因此来自VPN 的数据可以被信赖。在本例中，管理员先登录 VPN 服务器，然后使用企业内部网络管理应用服务器，这个过程与在企业内部网络中管理应用服务器一样是安全的。

从上面这个简单的示例中，可以看出路由管理的重要性，因此大部分读者都需要掌握此部分内容。本节将介绍 Linux 系统中的路由管理命令 route。

10.2.1　查看系统中的路由表

在 Linux 系统中，如果主机连接了多个网络，这时管理员就需要关注系统中的路由表了。虽然主机只连接到一个网络时，系统中也存在路由表，但这意义不大，因为除直接与主机相连的网络外（这种网络称为直连网络），其他都由默认路由处理了。本小节将简单介绍如何查看路由表。

【路由表】

路由表是计算机或路由器用来存储路由信息的表，表中包含了许多到达不同网络的路由条目。每个路由条目包含的主要信息有目标网络（又包括目标网络号和目标网络的子网掩码）、路由代价、下一跳路由器地址等。

系统转发数据包时，会查询路由条目的目标网络，并将数据包交给到达目标网络的下一跳路由器。如果有多个可以到达目标网络的路由条目，系统将使用路由代价最小的路由条目。如果没有找到目标网络的路由条目，系统会将数据包交给默认路由指定的下一跳路由器。

此处仅简单介绍在默认情况下，系统如何使用路由表工作。关于路由表的更多说明和帮助可以参考网络方面的相关书籍。

🔔小知识：默认路由是指向默认网关的路由条目，默认路由和默认网关本质是相同的，因此初学者可以简单地认为默认路由就是默认网关。

【Linux 系统中的路由表】

（1）查看主机路由表可以使用命令 route：

```
#使用 route 命令查看系统中的路由表
# route
Kernel IP routing table
Destination Gateway         Genmask         Flags Metric  Ref   Use   Iface
192.168.118.0   *           255.255.255.0    U         0    0      0   eth0
192.168.204.0   *           255.255.255.0    U         0    0      0   eth1
169.254.0.0     *           255.255.0.0      U         0    0      0   eth1
default     192.168.118.1   0.0.0.0          UG        0    0      0   eth0
```

上面的示例命令输出了系统内核的路由表，这个路由表中的几个字段及含义如下。

- ❑ Destination：目标网络号，default 表示这是一条默认路由。
- ❑ Gateway：网关地址（即下一跳路由器地址），其中“*”表示该网络与主机直接

相连，不需要使用网关地址。

- Genmask：表示目标网络的子网掩码。
- Flags：路由标记。
- Mertic：表示此路由条目到达目标网络的代价值。路由代价值的数字越高，表示代价越大，关于代价值的计算读者需要参考网络方面的书籍。
- Ref：路由条目被引用的次数（Linux 内核没有使用这个字段）。
- Use：路由条目被其他路由软件查找的次数。
- Iface：表示到达目标网络对应的接口。

在上面的几个字段中，路由标记用于指示当前路由条目的状态。其常见的几个标记及含义如下。

- U：表示当前路由条目处于活动状态。
- H：目标是一个主机。
- G：指向默认网关的路由条目。
- R：恢复动态路由产生的条目。动态路由是指路由器根据网络情况自动产生路由条目的过程。
- D：后台路由程序动态地安装。
- M：由路由后台程序修改的条目。
- C：缓存的路由条目。
- !：拒绝路由的条目。

其中，只有当系统作为一个路由器时，才会出现 D、M、R 标记。由于本书不以网络作为主要内容，因此读者需要参考网络方面的书籍了解这些内容。

从本例的输出信息可以看出，当前有两个网络与系统直接相连，分别是 192.168.118.0 和 192.168.204.0，当前系统使用的默认网关地址为 192.168.118.1。

（2）有时可能希望让路由表中的网关地址显示得更直观一些，这时可以使用选项 n：

```
#选项 n 表示显示网关的具体地址
# route -n
Kernel IP routing table
#由于是直连网络，不需要使用网关，因此网关地址显示为 0.0.0.0
Destination Gateway         Genmask        Flags Metric   Ref    Use   Iface
192.168.118.0   0.0.0.0     255.255.255.0   U        0      0      0    eth0
192.168.204.0   0.0.0.0     255.255.255.0   U        0      0      0    eth1
169.254.0.0     0.0.0.0     255.255.0.0     U        0      0      0    eth1
default     192.168.118.1   0.0.0.0         UG       0      0      0    eth0
```

注意：在 Linux 内核路由中还有一个直连网络 169.254.0.0，这是微软公司保留的 B 类私有网络地址。一般情况下当 Windows 主机无法使用 DHCP 服务获得 IP 地址时，使用这个 B 类私有网络中的地址。

10.2.2　添加默认路由

如 10.1 节中所述，有时需要使用 ifconfig 命令配置网络接口，以便于分辨网络接口对

应的网络。如果还要添加默认网关，执行进一步测试，需要使用 route 命令，以添加默认路由的方式添加默认网关。本小节将简单介绍如何使用 route 命令添加默认路由。

【命令格式】

使用 route 命令设置默认路由的基本格式如下：

```
route [add|del] default gw ipaddress
```

其中 add 和 del 分别表示添加、删除路由条目，其后的 gw 表示这是一个默认路由，ipaddress 是默认路由的 IP 地址。

【用法示例】

在设置之前还应该知道默认路由使用的具体地址。默认路由的地址通常应该是路由器的一个端口的 IP 地址，或者是安装有路由功能的计算机的 IP 地址。

（1）例如要设置默认路由地址为 192.168.204.1：

```
#使用 route 命令添加默认路由
#使用 default gw 表示这是一条默认路由
# route add default gw 192.168.204.1
```

使用以上命令添加默认路由时，一定要确保默认路由使用的地址处于主机的直连网络中。配置完成之后，就可以使用 ping 命令测试了。

（2）如果需要删除一个默认路由，可以使用如下命令：

```
#使用 route 命令删除默认路由
#使用 del default gw 表示删除一条默认路由
# route del default gw 192.168.204.1
```

使用命令方式配置 IP 地址、默认网关可以快速确定网络接口对应的网络。这种方式虽然不能保存配置信息，但可以快速确定网络接口对应的网络，因此管理员们经常使用。

10.2.3　添加路由条目

许多时候一台服务器主机要同时连接到多个网络，例如一台 Web 服务器可能在连接到 Internet 的同时，还连接到内部网络。这时需要管理员配置服务器的路由信息，以便于内部网络中的不同部门、不同管理员访问。本小节将简单介绍如何添加路由条目。

【命令格式】

添加路由条目时命令 route 的基本格式如下：

```
route [add|del] [net|host] ipaddress1 [netmask mask] [gw|dev] ipaddress2
|device
```

【参数说明】

添加路由条目时，其可用的参数及含义如下。

- add、del：分别表示添加、删除一个路由条目，
- net、host：表示要添加的路由条目的目标是一个网络或主机。
- ipaddress1：表示目标网络或目的主机的 IP 地址。
- ipaddress2：下一跳路由器的 IP 地址。
- netmask：表示目标网络的子网掩码。当目标是一个主机时，此参数应该省略。

- ❑ gw、dev：分别用于指定下一跳路由器的地址或要使用的网络接口。

虽然这个命令的参数很多，但其使用方法与路由器的添加静态路由的方法基本一致。

提示：静态路由通常是管理员手动添加的路由条目，本节中介绍的路由条目大多属于此类。

【用法示例】

（1）使用 route 命令添加路由条目需要配合使用参数 add。与路由器一样，添加的路由条目指定的目标可以是一个网络，也可以是一个主机。

例如要添加一个到网络 192.168.205.0 的路由条目：

```
#使用参数 add 表示添加一个路由条目
#net、netmask 参数分别表示目标网络的网络号和子网掩码
#gw 参数用于指定下一跳路由器的 IP 地址
# route add -net 192.168.205.0 netmask 255.255.255.0 gw 192.168.204.1
# route
Kernel IP routing table
Destination    Gateway        Genmask        Flags Metric Ref    Use  Iface
192.168.118.0  *              255.255.255.0  U     0      0      0    eth0
#以下为新添加的路由条目
192.168.205.0  192.168.204.1  255.255.255.0  UG    0      0      0    eth1
192.168.204.0  *              255.255.255.0  U     0      0      0    eth1
default        192.168.118.1  0.0.0.0        UG    0      0      0    eth0
```

从上面的示例命令输出中可以看出，路由条目已经添加成功。由于此路由条目的存在，以后发送到网络 192.168.205.0 的数据都会交给 192.168.204.1 转发，而不再通过 192.168.118.1 转发。

（2）上面的命令相当长，而且输入子网掩码也比较麻烦，此时可以像 10.1 节中那样以位数表示子网掩码：

```
#route 命令也支持使用位数表示子网掩码
# route add -net 192.168.205.0/24 gw 192.168.204.1
```

可以使用 route 命令进行验证，此命令的功能与前一条示例命令是一样的。

（3）有时需要指定主机的路由，这样做的目的可能是为了隔离访问（即只允许访问某个网段中的一台主机），这时可以使用参数 host 指定目标主机的 IP 地址。

例如要添加一条到主机 192.168.78.25 的路由条目：

```
#使用 host 参数表示添加到主机的路由条目
# route add -host 192.168.78.25 gw 192.168.204.1
#使用 route 命令验证
# route
Kernel IP routing table
Destination    Gateway        Genmask          Flags Metric Ref Use  Iface
192.168.78.25  192.168.204.1  255.255.255.255  UGH       0    0   0   eth1
192.168.118.0  *              255.255.255.0    U         0    0   0   eth0
……
```

添加了以上路由条目之后，就可以访问 IP 地址为 192.168.78.25 的主机了。但可能无

法访问主机所在网络的其他主机，这取决于默认网关中是否有相应的路由条目。

注意： 在添加主机路由时，Linux 系统中的 route 命令不允许使用 32 位子网掩码表示一个主机，而在一些路由器上可以使用此方法。

（4）有时可能需要将 Linux 系统当作路由器来使用，这时也可以与路由器的静态路由那样，指定网络接口作为下一跳的地址。

例如指定到网络 192.168.206.0 的下一跳设备：

```
#使用 dev 参数指定下一跳地址为设备 eth1
# route add -net 192.168.206.0/24 dev eth1
# route
Kernel IP routing table
Destination     Gateway         Genmask          Flags  Metric Ref  Use  Iface
192.168.78.25   192.168.204.1   255.255.255.255   UGH       0    0    0   eth1
192.168.118.0   *               255.255.255.0     U    0    0    0        eth0
192.168.205.0   192.168.204.1   255.255.255.0     UG        0    0    0   eth1
192.168.204.0   *               255.255.255.0     U         0    0    0   eth1
192.168.206.0   *               255.255.255.0     U         0    0    0   eth1
……
```

（5）许多时候，可能由于网络结构发生改变或者已经不再需要某个主机或网络，需要从路由表中删除一个路由条目。这时可以配合使用参数 del，删除一个已经存在的路由条目。

例如要删除到网络 192.168.206.0 的路由条目：

```
#使用参数 del 删除路由条目，并用 route 命令验证结果
# route del -net 192.168.206.0/24
# route
Kernel IP routing table
Destination       Gateway          Genmask         Flags Metric  Ref  Use  Iface
192.168.78.25   192.168.204.1   255.255.255.255  UGH      0      0   0    eth1
192.168.118.0   *               255.255.255.0    U        0      0   0    eth0
192.168.205.0   192.168.204.1   255.255.255.0    UG       0      0   0    eth1
192.168.204.0   *               255.255.255.0    U        0      0   0    eth1
169.254.0.0  *                  255.255.0.0      U        0      0   0    eth1
default      192.168.118.1      0.0.0.0          UG       0      0   0    eth0
```

从上面的示例命令输出中可以看出，到网络 192.168.206.0 的路由条目已经被删除。

（6）route 命令虽然可以添加路由条目，但当系统重新启动时，添加的路由条目却不能保存。为使每次系统重启后都能自动运行 route 命令添加路由条目，可以将相应的命令加入到文件/etc/rc.d/rc.local 内（系统启动时将自动运行该文件中的内容）。

例如要在系统启动时，自动添加到网络 192.168.206.0 和 192.168.205.0 的路由条目，可以修改/etc/rc.d/rc.local 为如下内容：

```
#使用 cat 命令查看文件 rc.local 的内容
# cat /etc/rc.d/rc.local
#!/bin/sh
#
……
touch /var/lock/subsys/local
```

```
#以下为新添加的内容
#add IP routing
route -net 192.168.205.1/24 gw 192.168.204.1
route -net 192.168.206.1/24 gw 192.168.204.1
```

（7）如果有多个路由条目，使用以上方法添加将是一件非常麻烦的事情。虽然可以不厌其烦地添加多个路由条目，但随着路由条目的增加，系统性能会下降，因此应该规划并使用路由汇总功能。

注意：路由汇总是指将一组路由汇聚成单个路由广播的过程。路由汇总的好处有：可以减少路由查询时间、减小路由表尺寸等。关于路由汇总功能的实现，感兴趣的读者可以参考相关网络书籍中的可变长子网掩码等内容。

例如使用路由汇总功能将 192.168.192.0 至 192.168.223.0 的网络汇聚成一个路由广播并添加到路由表中：

```
#使用 192.168.192.0/19 表示一组路由
# route add -net 192.168.192.0/19 gw 192.168.204.1
# route
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref Use Iface
192.168.78.25   192.168.204.1    255.255.255.255 UGH      0     0   0   eth1
192.168.118.0   *                255.255.255.0   U        0     0   0   eth0
192.168.205.0   192.168.204.1    255.255.255.0   UG       0     0   0   eth1
192.168.204.0   *                255.255.255.0   U        0     0   0   eth1
192.168.192.0   192.168.204.1    255.255.224.0   UG       0     0   0   eth1
……
```

10.3　主机名称命令 hostname

在 Linux 系统中，主机名称（有时也称主机名、计算机名）的全称一般分为两部分，第 1 部分用于标识主机，第 2 部分用于标识主机系统所在域。例如一个合法的主机名：web.example.com，可以看出，主机名称实际上就是一个域名。的确如此，早期主机名的作用就是快速访问网络中的计算机，但现在管理员主要用来分辨主机。

例如网络中有 5 台 Web 服务器，其计算机名分别为 Web1 至 Web5。如果同时连接到这 5 个服务器，这时管理员通常通过主机名快速分辨当前正在输入命令的服务器。这是因为命令提示符会显示主机名称。

如果有多个主机需要管理，不妨使用上面的方法，以免在错误的主机中，错误地执行了管理命令（例如错误地执行了 rm、mkfs 等，这可能是致命的错误）。除此之外，主机名在某些服务中也有重要的用途（例如 Samba 服务器等），但这已经超出本书的讨论范围，读者可以阅读相关文档。主机名称管理主要使用命令 hostname，本节将简单介绍该命令。

10.3.1　查看主机名称

虽然命令提示符中显示了主机名称，但有时我们希望能看到当前主机名所在的域，这

时可以直接使用命令 hostname。

（1）使用 hostname 命令时，不需要使用任何选项和参数：

```
#使用 hostname 命令显示主机名的全称
[root@localhost ~]# hostname
localhost.localdomain
```

上面的示例命令输出了主机名称，第 1 部分 localhost 表示当前主机的名称，第 2 部分 localdomain 表示当前系统所在的域。命令输出的是 RHEL5.3 中默认的主机名称，使用其他系统和 Shell 时，命令提示符和主机名可能会不同。

（2）主机名称通常存放在系统环境变量 HOSTNAME 中，因此也可以使用查看环境变量的方法查看主机名：

```
#使用查看环境变量的方法查看主机名称
#注意变量名应该使用大写
[root@localhost ~]# echo $HOSTNAME
localhost.localdomain
```

10.3.2　修改主机名称

在 Linux 系统中，修改主机名称有用命令修改和修改配置文件两种方法。使用命令修改主机名通常用于测试主机名是否可用，这种方法修改的主机名在系统重启后将丢失。本小节将简单介绍如何使用这两种方法修改主机名。

说明： 除了本小节中介绍的方法外，还可以使用 setup 工具集的方法修改主机名。setup 工具集的设置方法，可以参考 10.1.4 小节的相关内容，此处不再赘述。

1．使用命令修改主机名

如果要使用命令临时修改主机名，仍然应该使用 hostname 命令。例如：

```
#使用 hostname 命令修改主机名
[root@localhost ~]# hostname Web1.example.com
[root@localhost ~]# hostname
Web1.example.com
```

从上面的示例命令输出可以看出，使用 hostname 命令设置的主机名称将会立即生效，但由于当前系统已经处于登录状态，因此命令提示符没有发生变化。

此时如果重新登录系统，就会看到命令提示符变为：

```
[root@Web1 ~]#
```

使用命令修改主机名后，如果系统重新启动，这个设置将会丢失。如果需要保存修改的主机名称，就要使用修改配置文件的方法。

2．使用配置文件修改主机名

在 RHEL5.3 中，主机名保存在文件/etc/sysconfig/network 中。但如果只修改这个文件，

系统中依赖主机名的服务和软件将有可能无法正常工作，例如 sendmail（该服务主要用于向系统中的其他用户发送邮件），因此还需要修改 hosts 文件。此处简单介绍如何修改这两个文件。

查看主机名称配置文件的内容如下：

```
#使用 cat 命令查看文件/etc/sysconfig/network 的内容
# cat /etc/sysconfig/network
#第 1 行表示是否需要开启系统网络网络（默认为 IPv4）
NETWORKING=yes
#第 2 行表示是否需要开启 IPv6 支持
NETWORKING IPV6=no
#第 3 行表示当前系统使用的主机名
HOSTNAME=localhost.localdomain
```

这个文件的内容非常简洁，上面已经给出了详细的说明，读者按需要修改这个文件即可。由于配置文件的第 3 行中设置了主机名，因此本示例中只需要修改该行的内容即可。

在本例中要将主机名设置为 Web1.example.com，可以将文件内容修改为如下内容：

```
#使用 cat 命令查看修改后的配置文件内容
# cat /etc/sysconfig/network
NETWORKING=yes
NETWORKING IPV6=no
#注意此处使用了“#”将以前的设置注释掉，这样做的好处是可以随时恢复这些设置
#HOSTNAME=localhost.localdomain
HOSTNAME=Web1.example.com
```

在上面的示例配置文件中，使用“#”注释掉原来的内容，通常建议设置其他配置文件时，也使用这种办法以便于出问题时恢复。

提示： 如果将 NETWORKING 和 NETWORKING_IPV6 都设置为 no，系统将不会开启任何网络支持。如果系统启动时已经默认关闭网络功能，要开启网络功能，可以修改这两个配置项，并手动启动系统网络服务 network。

设置完成后并不会立即生效，这时可以重新启动系统网络服务来让这些设置生效：

```
#重新启动系统网络服务，并用 hostname 命令验证
# service network restart
Shutting down interface eth0:                            [  OK  ]
……
Bringing up interface eth1:                              [  OK  ]
# hostname
Web1.example.com
```

从上面的命令输出中可以看到设置已经生效，这时只需要重新登录就可以看到命令提示符的变化。

3．修改hosts文件

要保证系统中依赖于主机名工作的服务和软件正常工作，仅仅修改主机名称配置文件是不够的，这时还应该连同 hosts 文件一并修改。hosts 文件用于保存计算机网络中主机名

和 IP 地址的对应情况，这个文件早期用于保存主机名和 IP 地址的对应关系，现在已经被 DNS 服务器替代。

hosts 文件普遍存在于需要联网的操作系统中，例如 Linux、Windows、UNIX、Symbian（Symbian，通常称为塞班，是一个用于智能手机的操作系统）等都包含有该文件。

（1）不同的操作系统 hosts 文件的位置不同。在 Linux 系统中，hosts 文件通常位于/etc 目录中：

```
#查看 hosts 文件的内容
# cat /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
#以下是 IPv4 和 IPv6 的环回网络与 IP 地址的映射关系
#记录的格式为：IP 地址 主机名.主机名所属域名  主机名
127.0.0.1             localhost.localdomain localhost
::1           localhost6.localdomain6 localhost6
```

在 hosts 文件中，定义了主机名和 IP 地址之间的映射关系。如果要添加一个域名与 IP 地址之间的映射关系，只需要向其中写入一条记录即可。

（2）在本例中，可以修改 hosts 文件的内容如下：

```
#使用 cat 命令查看修改后的 hosts 内容
# cat /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1             localhost.localdomain localhost
::1           localhost6.localdomain6 localhost6
#以下为新添加的内容
127.0.0.1             Web1.example.com Web1
```

（3）修改完成后将会立即生效，此时可以使用 ping 命令检查其是否连通：

```
#使用 ping 命令验证 hosts 文件修改是否成功
# ping Web1.example.com
PING Web1.example.com (127.0.0.1) 56(84) bytes of data.
64  bytes  from  localhost.localdomain  (127.0.0.1):  icmp_seq=1  ttl=64
time=0.140 ms
64  bytes  from  localhost.localdomain  (127.0.0.1):  icmp_seq=2  ttl=64
time=0.111 ms
64  bytes  from  localhost.localdomain  (127.0.0.1):  icmp_seq=3  ttl=64
time=0.093 ms

--- Web1.example.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.093/0.114/0.140/0.022 ms
```

关于 ping 命令的更多应用，将在 10.5 节中进行介绍。

10.4 设置网络冗余

许多企业中都有关键性业务，这些服务一刻也不能中止，例如数据库、Web 服务等。

设计这些服务的结构时，除了电源冗余（有些服务器有多个冗余电源）、系统冗余（使用负载均衡、多服务器同时提供服务等）和存储器冗余（使用带冗余功能的存储器）是不够的。如果网络出现问题，这些关键性服务同样会中断。

从网络角度讲，冗余网络包含网络层冗余和链路层冗余两部分，使用的方法有热备路由（防止网关失效的机制）、快速生成树（防止链路失效的机制）等。但这些都不是本书讨论的内容，此处仅讨论系统本身如何做到网络冗余（即系统与交换机的连接冗余）。除此之外，还将讨论如何在系统中实现网络层面的负载均衡（即将系统的网络传输压力分散到多个网络连接中）。

由于网络冗余是用在服务器中的一项比较实用的技术，读者可以按自身的需要选择是否学习本节中的内容。

小知识： 冗余和负载均衡是服务器、运营领域的常见技术。冗余是指重复的系统部件，当主要系统部件发生故障时，冗余部件将替代其进行工作。可以将冗余理解为允许出错的机制（通常称为容错机制）。负载均衡是指将工作均衡地分布到多个处理单元的技术，通常负载均衡是建立在网络的基础上的。例如使用负载均衡技术将对某个网站的访问压力，分散到两个 Web 服务器上。

10.4.1　bonding 简介

提到网络层面的负载均衡，可能许多了解 CISCO（Cisco Systems, Inc，思科系统公司，全球第一大网络设备制造商）网络技术的读者，会想起思科的以太网通道聚合技术（Ethernet Channel）。以太网通道聚合技术可以让多个不同的网络端口聚合成一个端口，以此方法来提供更大的网络带宽。在 Linux 系统中，也提供了非常类似但功能更加强大的 bonding 技术，此处将简单介绍 bonding 技术。

面对日益复杂的网络应用环境，原来单一的网络接口技术在业务量巨大的服务器上备受关注，这些单一接口日益成为系统服务的瓶颈。为此许多公司推出了将多个物理通道聚合为一个端口的技术，例如 CISCO 公司的 Ethernet Channel，华为的端口汇聚（与以太网通道聚合技术类似）等，Linux 系统在 2.4 版（内核版本）中就提供了 bonding 技术。与网络设备制造商提供的多端口绑定技术不同，bonding 不仅可以将多个不同的网络连接绑定在一起实现负载均衡模式，还可以将多个网络连接设置为备份连接实现网络连接冗余模式。bonding 在企业中的应用如图 10.6 所示。

当 bonding 工作在负载均衡模式时，会将多个网络接口绑定为一个网络接口，这些网络接口使用相同的 IP 地址，以及相同的 MAC 地址（有些模式下 MAC 地址可能会不同）。当其发送数据时会通过不同的算法（按其使用的模式不同）将这些数据通过不同的网络接口发送出去。在图 10.6 中，两个服务器同时与两个交换机相连，发送和接收的数据由两个交换机分担。这样就可以减轻大量的数据包对单个网络接口的压力，也可以增加服务器的可用带宽。

当 bonding 工作在冗余模式时，会将多个网络连接设置为备份连接。当主要网络连接断开后，备份连接将会自动启动，并替代主要连接继续工作。由于这个过程发生在底层，因此业务的使用者将不会察觉到这个过程有任何异常。在图 10.6 所示的这个应用模型中，

每个服务器都拥有两个链路，即使有一个链路断开，服务器也可以使用备用链路传输数据。

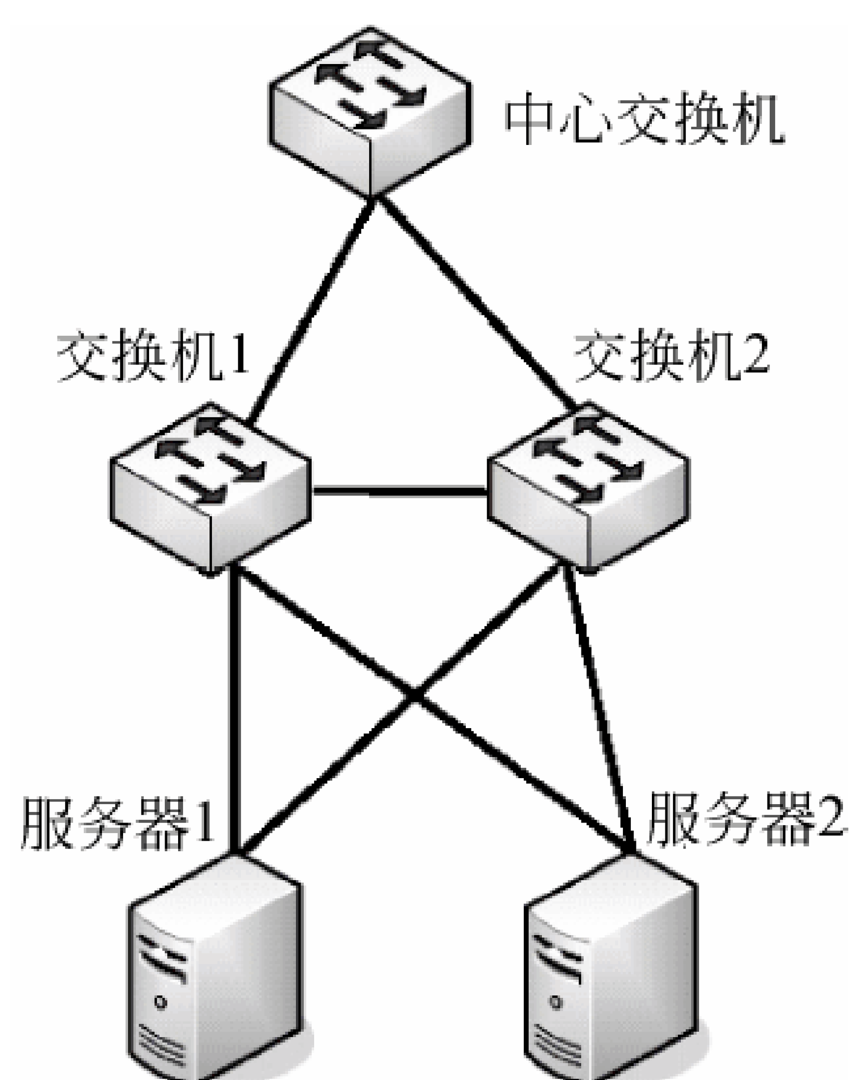


图 10.6　bonding 应用模型

Linux 系统中的 bonding 是一种非常实用的技术，不仅可以实现网络数据的负载均衡，还可以实现网络链路冗余设置，这在许多大中型企业的服务器上使用得非常普遍。

10.4.2　bonding 的模式

利用 Linux 系统中的 bonding 技术，管理员可以实现网络连接负载均衡，也可以实现网络连接冗余设置。决定 bonding 工作的最重要因素是 bonding 的模式。在学习配置 bonding 之前，应该了解其几种基本工作模式的基本功能和实现方法，以便于按实际需要进行选择。

bonding 的工作模式和对应的功能如下。

- Round-robin policy 或数字 0：轮转策略模式。这种模式会将要发送的数据平均地从所有绑定的网络接口中发送出去，此时可以提供负载均衡和网络冗余功能。
- Active-backup policy 或数字 1：通常将这种模式称为主备模式。正常状态下只使用主要接口发送和接收数据，当主要接口断开后，bonding 将立即启动备用接口。这种模式只能提供网络冗余功能。
- XOR policy 或数字 2：基于地址的异或算法。这是一种由源地址、目标地址和接口数量共同决定数据包发送接口的算法。这种模式可以提供负载均衡和网络冗余功能。
- Broadcast 或数字 3：将所有要发送的数据从所有网络连接上发送出去（非常类似于广播），这种模式只能提供网络冗余功能。
- 802.3ad 或数字 4：使用 802.3ad 协议将所有网络连接进行聚合，通常这种方式需要交换机支持。这是一种不常用的模式，此处不作过多介绍，具体信息可以查看协议内容。
- Balance-tlb 或数字 5：自适应负载均衡模式，系统将会自动测定发送和接收使用的接口和速度。
- Balance-alb 或数字 6：使用 ARP 协议控制的负载均衡模式。这种方式不需要特殊的交换机支持。

在 bonding 的所有模式中，0、1 和 6 最为常见，也是最经常使用的模式。在构建一个负载均衡或网络冗余之前应该了解这些模式，并按自身使用的交换机的实际情况选择合适的模式（如果交换机不支持，可能会导致交换机 MAC 地址表频繁抖动，影响网络性能，详细信息请参看与此有关的网络方面的书籍）。

10.4.3　设置网络接口

从本小节开始，将以一个示例讲解如何实现 bonding。在执行所有配置之前，应该先设置相关接口，以支持 bonding。在本例中以网络接口 eth0 和 eth1 为例。

注意：并不是所有网卡都支持 bonding 技术，该技术通常只能应用于拥有 BIOS 芯片（BIOS 芯片多用于从网络引导系统，服务器使用的千兆网卡大多带有 BIOS 芯片）的网卡。无此条件的读者，可以在虚拟机中学习本节内容。

（1）将两个网络接口的配置文件修改为以下内容：

```
#使用 cat 命令查看网络接口 eth0 的配置文件
# cat /etc/sysconfig/network-scripts/ifcfg-eth0
# Intel Corporation 82541GI Gigabit Ethernet Controller
DEVICE=eth0
BOOTPROTO=none
ONBOOT=yes
TYPE=Ethernet
#使用 cat 命令查看网络接口 eth1 的配置文件
# cat /etc/sysconfig/network-scripts/ifcfg-eth1
# Intel Corporation 82541PI Gigabit Ethernet Controller
DEVICE=eth1
……
#省略部分与 eth0 的配置文件相同
```

将两个网络接口都设置为开机启动，并将获取网络连接设置的方式配置为不使用 DHCP。

（2）除此之外，还应该为新网络接口创建一个配置文件。在本例中，要使用 bonding 技术创建一个新网络接口，并命名为 bond0。

此时需要为新网络接口创建配置文件/etc/sysconfig/network-scripts/ifcfg-bond0，并手动写入配置信息：

```
#使用 cat 命令查看新网络接口的配置文件内容
# cat /etc/sysconfig/network-scripts/ifcfg-bond0
DEVICE=bond0
BOOTPROTO=none
ONBOOT=yes
NETMASK=255.255.255.0
IPADDR=192.168.121.115
GATEWAY=192.168.121.1
TYPE=Ethernet
```

这是一个服务器上的某个接口的真实配置，配置文件中设置了新网络接口使用的 IP

地址、子网掩码和网关等信息。

10.4.4　加载模块生成新的网络连接

由于 RHEL5.3 中默认并没有加载 bonding 模块，因此在使用 bonding 设置负载均衡和网络冗余之前，必须加载 bonding 模块。本小节将简单介绍如何加载 bonding 模块并生成新的网络接口。

小知识： 内核模块简称模块，是一种可以动态加载的程序。通过内核模块可以扩展内核的功能，通常用于设备驱动、文件系统等。可用的内核模块通常放在/lib/modules目录中。

（1）加载模块相关设置通常都写在配置文件/etc/modprobe.conf 中。查看该文件的内容如下：

```
#使用 cat 命令查看配置文件
# cat /etc/modprobe.conf
alias scsi_hostadapter pata_pdc2027x
alias eth0 e1000
alias scsi_hostadapter1 ata_piix
alias eth1 e1000
```

默认情况下，配置文件中设置了系统中加载的模块和使用的设备别名。

提示： 在某些发行版中（通常是一些较老的发行版），加载内核模块的文件可能是/etc/modules.conf。

（2）在文件最后加入新模块的加载内容，修改后的内容如下：

```
#使用 cat 命令查看修改后的配置文件
# cat /etc/modprobe.conf
alias scsi hostadapter pata pdc2027x
alias eth0 e1000
alias scsi hostadapter1 ata piix
alias eth1 e1000

#以下为新加入的内容
#第 1 行为 bonding 模块定义了一个别名 bond0（就是新网络接口的名称）
alias bond0 bonding
#第 2 行为 bond0 使用的选项和参数
options bond0 miimon=100 mode=0
```

在本示例中，mode=0 表示 bonding 使用的模式为轮转策略模式，可以按自身需求设置使用的模式。miimon=100 表示每隔 100 毫秒检测一次物理链路，如果检测到网络接口故障，bonding 将会自动切换网络接口和数据发送模式。

（3）完成模块加载之后，就可以重新启动网络服务让这个模块加载设置生效了：

```
#使用 service 命令重启系统网络服务
# service network restart
```

命令成功执行后，可以看到新网络接口 bond0 已经开启，但还不能使用该接口发送和接收数据，因为系统还不知道新接口中绑定了哪些网络接口。

（4）设置新接口的绑定信息，在本例中应该使用如下命令：

```
#使用 ifenslave 为新接口设置绑定信息
# ifenslave bond0 eth0 eth1
```

如果未返回任何错误，就表明已经成功将 eth0 和 eth1 绑定到新接口 bond0 中。使用 ifenslave 命令时需要注意，如果使用主备模式（即模式 1），第 1 个网络接口将作为主要接口。

（5）重新启用网络接口 bond0 即可使所有设置生效：

```
#使用 ifdown 和 ifup 重新启用网络接口 bond0
# ifdown bond0
# ifup bond0
```

这时新的网络接口 bond0 就已经可以接收和发送数据了。

（6）虽然已经将模块加载写入配置文件，但由于没有将网络接口绑定情况写入配置文件，因此重新启动之后这个接口将不能使用。为此可以将新接口绑定命令写入文件/etc/rc.local，以便于系统启动时自动绑定网络接口。

修改文件/etc/rc.local 的内容如下：

```
#查看文件/etc/rc.local 的内容
#如果无法区分文件/etc/rc.local 与/etc/rc.d/rc.local，就使用 ls -l 查看这两个文件
吧
# cat /etc/rc.local
#!/bin/sh
#
……
touch /var/lock/subsys/local

#以下为新写入的内容
ifenslave bond0 eth0 eth1
```

修改完成，如果系统重新启动，也可以使用 bonding 创建的网络接口 bond0。

10.4.5　验证设置

完成上一小节的设置之后，就可以使用命令 ifconfig 命令查看这几个接口的状态了，以验证 bond0 是否按照预计的情况工作。

（1）使用 ifconfig 命令查看所有网络接口信息：

```
#使用 ifconfig 命令查看网络接口信息
# ifconfig
#注意以下 3 个网络接口的 MAC 地址、收发包的统计信息、收发字节数的统计信息
bond0     Link encap:Ethernet  HWaddr 00:A0:D1:E2:17:86
          inet addr:192.168.111.105  Bcast:192.168.111.255  Mask:255.255.255.0
          inet6 addr: fe80::2a0:d1ff:fee2:1786/64 Scope:Link
          UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
```

```
          RX packets:658 errors:0 dropped:0 overruns:0 frame:0
          TX packets:207 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:58146 (56.7 KiB)  TX bytes:22023 (21.5 KiB)

eth0      Link encap:Ethernet  HWaddr 00:A0:D1:E2:17:86
          UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
          RX packets:326 errors:0 dropped:0 overruns:0 frame:0
          TX packets:107 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:28301 (27.6 KiB)  TX bytes:11350 (11.0 KiB)
          Base address:0xdc00 Memory:fea80000-feaa0000

eth1      Link encap:Ethernet  HWaddr 00:A0:D1:E2:17:86
          UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
          RX packets:332 errors:0 dropped:0 overruns:0 frame:0
          TX packets:100 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:29845 (29.1 KiB)  TX bytes:10673 (10.4 KiB)
          Base address:0xd880 Memory:fea40000-fea60000
……
```

从上面的命令输出中可以看到，网络接口 eth0 和 eth1 共同分担了 bond0 的数据流量。由于使用的是模式 1，因此 3 个接口都使用同样的 MAC 地址。

提示： 如果设置好 IP 地址等信息，此时可以使用 ping 命令测试网络接口 bond0 是否可用。

（2）除了使用 ifconfig 命令之外，还可以用查看/proc/net/bonding/bond0 的方法获取 bond0 的状态：

```
#通过查看/proc/net/bonding/bond0 的方法验证 bond0 是否按预计的目标工作
# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.4.0 (October 7, 2008)

#注意以下的状态信息
Bonding Mode: load balancing (round-robin)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: eth0
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:a0:d1:e2:17:86

Slave Interface: eth1
MII Status: up
Link Failure Count: 0
Permanent HW addr: 00:1b:21:54:19:19
```

从上面的命令输出中可以看到，相应的网络接口已经处于工作状态了。

本节仅以模式 0（轮转模式）为例，讲解如何使用 bonding 设置网络负载均衡和网络冗余，读者可以按自身的实际情况选择使用的模式。

10.5　网 络 工 具

许多 Linux 系统都不是孤立地存在于某一个网络中，它总是会与其他服务器或网络产生“交流”。当这种“交流”出现故障时，初学者可能会感觉比较难处理。即使是有经验的管理员，面对大型网络中的小故障可能都会感到十分困难，原因在于有许多故障点需要花时间去逐一排查。

为了解决网络故障，Linux 系统自带了许多用于排查各类故障的工具。利用这些工具，用户可以诊断网络的各个方面，排除网络故障。本节将简单介绍这些工具的使用方法，以及排除系统网络故障的基本思路。

10.5.1　测试连通命令 ping

ping 命令是系统管理员最常用的工具之一，其主要功能是检查目标主机与本地之机之间的双向连通性。双向连通性是指目标主机与本地主机之间能够双向收发数据。

【命令格式】

```
ping [option] ipaddress
```

在上面的格式中，ipaddress 可以是目标主机的 IP 地址、域名、主机名。如果使用域名、主机名作为目标主机地址，需要使用本地 hosts 文件 DNS 服务器解析。

【常用选项】

- ❑ c：指定发送数据包的个数。
- ❑ f：快速发送数据包，用于对当前主机与目标主机间的网络执行极限测试。
- ❑ s：指定发送的数据包大小。该选项主要用于测试链路质量和主机响应速度。
- ❑ I：指定发送测试数据包的网络接口。
- ❑ R：显示数据包经过的路由过程。
- ❑ t：指定数据包使用的 TTL 值（TTL 用于控制数据包的生命周期）。

除以上选项之外，ping 命令还有一些不常用的选项，读者可以阅读相关文档和帮助文件。

【用法示例】

ping 命令通过向目标主机发送 ICMP（Internet Control Message Protocol，Internet 控制报文协议）控制消息的方法测试两个主机间的连通性。由于某些主机或防火墙可能会禁止使用 ICMP 消息，因此可能会出现两台主机可以通信，但 ping 命令显示不通的情况。

（1）例如要测试与默认网关之间的连通性：

```
#使用 ping 命令测试与 192.168.204.1 的连通性
# ping 192.168.204.1
PING 192.168.204.1 (192.168.204.1) 56(84) bytes of data.
#以下为对方主机回应的情况
#其中最重要的是 time 的值，该值用于表示与目标主机间的网络延时
```

```
#延时越小表明响应速度越快
64 bytes from 192.168.204.1: icmp seq=1 ttl=255 time=0.803 ms
64 bytes from 192.168.204.1: icmp seq=2 ttl=255 time=0.828 ms
64 bytes from 192.168.204.1: icmp seq=3 ttl=255 time=0.832 ms
64 bytes from 192.168.204.1: icmp seq=4 ttl=255 time=0.757 ms

#此时按下快捷键 Ctrl+C 中断测试
--- 192.168.204.1 ping statistics ---
#以下为发送接收数据包的统计情况
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
#以下为网络延时的统计情况
rtt min/avg/max/mdev = 0.757/0.805/0.832/0.029 ms
```

与 Windows 系统中的 ping 命令不同，Linux 系统中的 ping 命令会持续发送数据包测试与目标主机间的连通性。要中断其测试，可以使用快捷键 Ctrl+C。

上面的示例命令一共发送了 4 个数据包，网关一一回应了这 4 个数据包。命令最后还输出了数据包的统计信息，包括发送数据包的总数、丢包率、最大延时和最小延时等信息，这些信息非常有助于管理员了解当前的网络状态。

提示：与网络传输速度一样，网络延时也是判断网络好坏的标准，但许多初学者容易混淆二者的概念，因此应该特别注意。

TTL 值是数据包的生存周期的体现，一个数据包发出时将会附带一个 TTL 值（上面的示例中是 255）。数据包每经过一个路由器，TTL 值就会减少 1。当 TTL 值为 0 时，路由器就会返回数据包不可达的消息。网络层使用这种特殊的方式防止路由环路，即一个数据无休止地被转发下去。不同的系统会使用不同的 TTL 值，常见的 TTL 值有 64、128 和 255 等，通过 TTL 值的方式可以判断数据包经过了几个路由器。

（2）有时可能要对网络连接做丢包率评估，需要指定要测试的数据包个数，这时可以配合使用选项 c。

例如要发送 3 个数据包测试与网关之间的通信情况：

```
#使用选项 c 指定发送测试数据包的个数
# ping -c 3 192.168.204.1
PING 192.168.204.1 (192.168.204.1) 56(84) bytes of data.
64 bytes from 192.168.204.1: icmp_seq=1 ttl=255 time=0.719 ms
64 bytes from 192.168.204.1: icmp_seq=2 ttl=255 time=0.824 ms
64 bytes from 192.168.204.1: icmp_seq=3 ttl=255 time=1.28 ms

--- 192.168.204.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.719/0.942/1.285/0.248 ms
```

使用上面的示例命令时，不需要使用快捷键中止，ping 命令只发送了 3 个数据就输出了统计信息。由于此处仅演示其用法，因此只发送了 3 个测试数据包，在实际情况中，为了获得更准确的丢包率，发送的数据包可能会更多。

（3）有时需要对目标主机之间的通信进行极限测试，即短时间内大量数据包的传输测试，可以配合使用选项 f。

例如对网关进行极限测试：

```
#使用选项 f 对 192.168.204.1 执行极限测试
# ping -f 192.168.204.1
PING 192.168.204.1 (192.168.204.1) 56(84) bytes of data.
....
#此时按下 Ctrl+C 键
--- 192.168.204.1 ping statistics ---
2237 packets transmitted, 2233 received, 0% packet loss, time 2198ms
rtt min/avg/max/mdev = 0.247/1.491/203.982/9.930 ms, pipe 14, ipg/ewma
0.983/63.743 ms
```

这个命令的输出与前面不一样，由于会在短时间内发送大量的数据包（大约每秒钟发送的数据在 1000～2000 之间），因此没有返回每个数据包的详情，而是使用点号“.”表示丢失了一个数据包。从命令的统计信息可以看出，ping 命令一共发送了 2237 个数据包，其中有 4 个数据包丢失了。

注意：此处的极限测试并非测试目标主机处理数据的能力，而是在短时间内向其发送大量的数据包，以此方式测试网络对突发数据包的处理能力。

（4）使用选项 f 执行极限测试时，如果不加以限制，可能会对网络造成影响。因此通常都将其与选项 c 一起使用：

```
#使用选项 f 执行权限测试时，使用选项 c 指定发送数据包的个数为 2000 个
# ping -f -c 2000 192.168.204.1
```

（5）默认情况下，ping 命令发送的数据包大小为 64 字节。如果测试网络和目标主机对大数据包的处理能力，可以配合使用选项 s 指定发送的数据包大小。

例如要向网关发送大小为 10 000 字节的测试数据包：

```
#使用选项 s 指定测试数据包的大小
# ping -s 10000 192.168.204.1
PING 192.168.204.1 (192.168.204.1) 10000(10028) bytes of data.
10008 bytes from 192.168.204.1: icmp seq=2 ttl=255 time=17.5 ms
10008 bytes from 192.168.204.1: icmp seq=3 ttl=255 time=17.4 ms
……
--- 192.168.204.1 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4002ms
rtt min/avg/max/mdev = 17.470/17.524/17.582/0.107 ms
```

从返回的结果可以看出，延时明显增大了。如果许多台计算机使用这种方式测试同一目标主机，可能会导致目标主机处理缓慢，甚至死机。

（6）许多时候当前主机上有多个网络接口，可能希望指定接口测试（指定接口测试一般用于指定数据包经过的路径）。此时可以配合使用选项 I 指定 ping 命令使用的网络接口。

例如指定测试网络接口 eth0 与目标主机之间的连通性：

```
#使用选项 I 指定测试用的网络接口为 eth0
# ping -I eth0 192.168.111.1
PING 192.168.111.1 (192.168.111.1) from 192.168.118.234 eth0: 56(84) bytes
of data.
64 bytes from 192.168.111.1: icmp seq=1 ttl=255 time=0.935 ms
64 bytes from 192.168.111.1: icmp_seq=2 ttl=255 time=2.25 ms
```

```
……
--- 192.168.111.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.935/1.402/2.251/0.506 ms
```

从上面返回的信息可以看出，当前主机与目标主机可以双向通信。

提示：通常只有当前系统为路由器时才使用选项 I，通常情况下由于默认路由的存在，其意义并不是很大。

（7）有时可能需要验证数据包经过的路径（多用于测试多个路由器上的路由项是否正确），这时可以配合使用选项 R 显示数据包经过的路径。

例如要测试与目标主机的连通性，并且显示数据包经过路径：

```
#使用选项 R 显示数据包经过的路径
# ping -R 192.168.111.10
PING 192.168.111.10 (192.168.111.10) 56(124) bytes of data.
64 bytes from 192.168.111.10: icmp seq=1 ttl=64 time=3.65 ms
#以下为数据包经过的路径
RR:    192.168.118.234
       192.168.111.1
       192.168.111.10
       192.168.111.10
       192.168.118.234

#以下为连通性测试结果
64 bytes from 192.168.111.10: icmp seq=2 ttl=64 time=1.03 ms    (same route)
64 bytes from 192.168.111.10: icmp seq=3 ttl=64 time=1.71 ms    (same route)
64 bytes from 192.168.111.10: icmp seq=4 ttl=64 time=5.37 ms    (same route)
64 bytes from 192.168.111.10: icmp seq=5 ttl=64 time=1.29 ms    (same route)

#以下为统计信息
--- 192.168.111.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 1.037/2.617/5.379/1.658 ms
```

命令先输出了一次测试结果，然后输出了数据包往返经过的路径（路径使用路由器地址表示）。

ping 命令获取路径的原理是从 1 开始累加 TTL 值，当其值减为 0 时，相应的路由器会返回生命值耗尽的消息，从而获取路径上的路由器 IP 地址。

（8）从前面的数据包生命周期中可以了解到，TTL 值可以用于返回路由器的地址。例如想要知道 ping 发出的数据包经过默认网关之后，第 4 个经过的路由器地址：

```
#使用选项 t 指定 TTL 值为 4
# ping -t 4 61.138.2.69
PING 61.138.2.69 (61.138.2.69) 56(84) bytes of data.
From 222.*.*.* icmp_seq=1 Time to live exceeded
From 222.*.*.* icmp_seq=2 Time to live exceeded
From 222.*.*.* icmp_seq=3 Time to live exceeded
```

```
--- 61.138.2.69 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2000ms
```

此处使用的 IP 地址是四川省 DNS 服务器的 IP 地址，第 4 个 IP 地址为 222.*.*.*（此处屏蔽掉原始 IP 地址）的路由器返回了生命耗尽的消息。

注意：ping 命令是管理员的常用命令，读者需要特别注意其用法。

10.5.2　网络路径测试命令 traceroute

许多时候主机处于一个庞大的网络中，主机到目标主机间的网络出现问题时（表现为 ping 命令返回超时），要找出故障节点可能会非常复杂，并且也非常麻烦。虽然使用“ping”命令也可以找出故障节点，但有时效果并不理想，这时可以使用网络路径测试命令 traceroute。

traceroute 命令的功能与 ping 命令的功能相似，也是用于测试与目标主机之间的连通性。不同的是，traceroute 命令会测试与目标主机间的路由器的连通性，即 traceroute 命令会测试通往目标主机间的所有节点的连通性，这样就可以很容易地找出本地系统主机与目标主机间的故障点，从而直接排查有问题的节点。

traceroute 命令使用目标主机的 IP 地址作为参数，测试并返回本地主机到目标主机间所有经过的路由器连通情况及测试结果。使用时不必使用任何选项。

例如要测试本地主机与目标主机间的所有路由器的连通情况：

```
#使用 traceroute 命令测试到目标主机 192.168.144.186 之间的所有路由器的连通情况
# traceroute 192.168.144.186
traceroute to 192.168.144.186 (192.168.144.186), 30 hops max, 40 byte packets
 1  192.168.118.1 (192.168.118.1)  4.916 ms  4.882 ms  5.252 ms
 2  192.168.144.186 (192.168.144.186)  0.822 ms  0.720 ms  0.595 ms
```

命令输出了所有经过的路由器的 IP 地址（包括目标主机）及测试产生的结果。由于需要逐一测试，因此 traceroute 命令的执行速度比 ping 命令要慢很多。

事实上 traceroute 命令与 ping 命令的选项 R 工作原理相同，都是利用 TTL 值获取中间路由的 IP 地址。不同的是 traceroute 命令将测试中间路由的连通性。

10.5.3　查看网络状态命令 netstat

netstat 命令通常用于查看系统中的网络连接状态，包括系统当前网络接口状态、路由表、系统中服务端口使用情况等。系统管理员们经常使用 netstat 命令查看系统中网络接口状态，以及系统与外部的连接情况，以便确认系统的安全性，例如是否有未授权用户使用终端登录服务器等。本小节将简单介绍 netstat 命令的使用方法。

【命令格式】

```
netstat [option]
```

使用 netstat 命令还需要一些前提知识，主要是 TCP、UDP 连接等方面的基础知识。

【常用选项】

- ❑ i：查看网络接口的使用情况。
- ❑ t：列出正在使用的 TCP 连接。
- ❑ u：列出正在使用的 UDP 连接。
- ❑ a：查看所有正在连接中的套接字。
- ❑ n：以 IP 地址的形式显示（而非域名）。
- ❑ p：显示套接字对应的进程名及 PID。
- ❑ l：只显示正处于监听状态的套接字。

提示： 套接字是系统处理相同端口的不同 TCP、UDP 连接的接口，读者可以阅读网络和 C 语言方面的书籍了解相关知识。

【用法示例】

（1）许多时候我们都使用前面介绍的 ifconfig 命令显示接口使用情况，但这个命令的显示结果非常繁杂。这时可以使用 netstat 命令的选项 i 查看网络接口的使用情况。

例如要查看系统中网络接口的使用情况，用如下命令：

```
#使用选项 i 查看网络接口的使用情况
# netstat -i
Kernel Interface table
Iface   MTU    Met  RX-OK  RX-ERR   X-DRP  RX-OVR  TX-OK  TX-ERR  TX-DRP  TX-OVR   Flg
eth0    1500   0    117932  0          0       0   34170    0       0       0     BMRU
eth1    1500   0    13621   0          0       0   4141     0       0       0     BMRU
lo      16436  0    58      0          0       0   58       0       0       0     LRU
```

上面这个示例命令的输出非常简单，Iface 表示网络接口，MTU 和 Met 表示最大传输单元及其度量值。RX-OK 和 TX-OK 表示发送和接收正常的数据包数量，RX-ERR 和 TX-ERR 表示发送和接收到错误的数据包的数量。RX-DRP 和 TX-DRP 表示发送和接收时，被丢弃的数据包的数量，RX-OVR 和 TX-OVR 表示发送和接收时溢出的数据包的数量。

命令输出的最后一个字段 Flg 标识了接口当前的状态，使用大写字母标识接口当前的状态信息。常见的字母及其含义如下。

- ❑ B：表明该接口设置了广播地址。
- ❑ M：表明接口处于混乱模式，即可以接收所有数据包。
- ❑ R：表示当前接口正在运行，可以接收数据包。
- ❑ U：表示当前接口已经处于开启状态。
- ❑ L：表明当前接口是一个环回接口。

如果需要查看网络接口是否正常工作，可以使用此命令并查看接口的 Flg 字段中是否包含 R 和 U 标记。

（2）有时管理员可能需要监测网络，查看哪些应用程序或服务正在使用系统，这时可以使用 netstat 命令的选项 t 和 u，查看当前正与系统连接的 TCP、UDP 连接。

例如要查看当前系统中正在使用的连接情况：

```
#使用选项 tu 显示当前系统的连接
# netstat -tu
Active Internet connections (w/o servers)
```

```
Proto Recv-Q Send-Q Local Address          Foreign Address          State
Tcp 0  132 ::ffff:192.168.118.234:ssh   ::ffff:192.168.144:index-net ESTABLISHED
```

从上面的输出中可以看出，当前正有一个来自网络 192.168.144.0 的 TCP 连接，该连接正在使用系统的 ssh 服务。

在上面的命令输出中，State 字段表示当前该连接的状态。常见的状态描述及其含义如下。

- ESTABLISHED：表示当前套接字已经存在一个连接。
- SYN_SENT：表示当前套接字正在积极尝试建立连接。
- SYN_RECV：表示已经收到一个连接的请求。
- CLOSE_WAIT：表示远程端口已经关闭，正在等待套接字关闭。
- FIN_WAIT1：表示套接字已经关闭，连接正在关闭。
- CLOSED：表示套接字没有被使用。
- LISTEN：表示套接字正处于监听状态。
- UNKNOWN：表示套接字的状态不可知。

提示： 上面列出的这些状态与 TCP、UDP 连接协议中定义的工作原理有关，如果读者需要了解这些内容，请参考 TCP、UDP 方面的书籍。

在上面列出的状态中，ESTABLISHED 和 LISTEN（监听状态）都应该引起管理员的足够重视，管理员应该仔细查看其中的可疑连接以保证系统安全。

（3）除了系统当前正在使用的连接，通常管理员还关心系统中有哪些端口正处于监听状态。监听状态就好比打开的“门”（端口），有些“门”是主人故意打开迎接客人的通道（为提供服务而打开的端口），但如果家里有个居心叵测的家伙打开了“后门”（可能是木马等后门程序打开的端口，也可能是具有安全漏洞的服务端口），迎接的可能就是小偷。

如果要查看所有端口及其状态，可以配合使用选项 a 查看所有正在连接中的套接字：

```
#使用选项 a 查看所有套接字
# netstat -atu
Active Internet connections (servers and established)
#LISTEN 状态表示打开的端口
Proto Recv-Q Send-Q Local Address             Foreign Address          State
tcp    0    0 localhost.localdomain:2208     *:*                      LISTEN
tcp    0    0 *:sunrpc                       *:*                      LISTEN
tcp    0    0 *:723                          *:*                      LISTEN
tcp    0    0 localhost.localdomain:smtp     *:*                      LISTEN
tcp    0    0 localhost.localdomain:2207     *:*                      LISTEN
tcp    0    0 *:ssh                          *:*                      LISTEN
tcp    0  264 ::ffff:192.168.118.234:ssh   ::ffff:192.168.4:agriserver ESTABLISHED
……
```

命令输出了系统中所有正在使用网络的端口和端口对应的协议等内容，通过查看这些信息，管理员可以了解系统网络的当前使用情况。

如果当前主机用于提供服务，建议将其他端口全部关闭，以免引起安全性问题。关闭其他端口需要将打开这些端口的服务、进程关闭。

（4）读者可能注意到上面的命令输出中，IP 地址看起来很不方便。要更加直观地查看当前系统中的网络连接情况，可以用选项 n，以 IP 地址的方式显示，用选项 p，显示套接字对应的进程和 PID：

```
#使用选项 t、u、n、p 以更直观的方式查看连接及进程情况
# netstat -tunp
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address      Foreign Address    State  PID/Program
name
tcp  0     148   ::ffff:192.168.118.234:22       ::ffff:192.168.144.186:3021
ESTABLISHED 19297/1
```

从上面的命令输出结果可以看出，当前系统有一个来自 192.168.144.186 的连接，使用的系统端口是 22（即 ssh 服务）。

（5）在查看了当前系统中的网络使用情况之后，管理员需要了解是哪些进程或服务正在使用这些连接或正在监听网络。这就好比主人查看了所有“门”的情况后，需要查看是哪些家伙打开并管理这些“门”，以便于从家里清除那些不怀好意的家伙。

这时可以配合使用选项 l 只显示正处于监听状态的套接字。例如要查看当前正在监听网络的进程及其 PID：

```
#使用选项 n、l、p 显示所有正在监听网络的进程及其 PID
# netstat -tunlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address  Foreign Address State PID/Program name
tcp      0       0    127.0.0.1:2208 0.0.0.0:*    LISTEN    3344/hpiod
tcp      0       0    0.0.0.0:111    0.0.0.0:*    LISTEN    3037/portmap
tcp      0       0    0.0.0.0:723    0.0.0.0:*    LISTEN    3085/rpc.statd
tcp      0       0    127.0.0.1:25   0.0.0.0:*    LISTEN    3446/sendmail:acce
tcp      0       0    127.0.0.1:2207 0.0.0.0:*    LISTEN    3349/python
tcp      0       0    :::22          :::*         LISTEN    3381/sshd
……
```

命令输出了当前正在监听网络的进程及其 PID，以及 IP 地址端口等信息。

查出了这些监听网络的进程和服务之后，就可以关闭不需要使用的服务和进程，也可以对一些可疑端口设置防火墙规则，以免系统安全受到威胁。

注意：netstat 命令是管理员常用的一个工具，可以很方便地监控网络使用情况。该命令还有许多用法，读者可以参考其他文档了解其详细使用说明。

10.5.4　域名解析工具 dig 和 nslookup

大部分情况下人们都会使用域名浏览网站，例如访问网易时用 www.163.com，新浪 www.sina.com.cn 等。当我们在浏览器中输入以上域名时，主机会首先与 DNS 服务器联系，要求 DNS 服务器解析域名对应的 IP，以便浏览器使用 IP 地址打开网站。

但有时这个解析过程会出现问题，管理员可能无法确认哪个过程发生了错误，这时就需要借助域名解析工具进行判断。本小节将简单介绍域名解析工具 dig 和 nslookup。

域名能被 DNS 服务解析必须有两个前提条件，首先是本地 hosts 文件正确无误。如果 hosts 文件中的记录将要访问的域名指向不存在的地址，那么就算 DNS 服务器能正常解析，也不能起作用。其原因是系统查询域名时，总是先查询本地 hosts 文件，因此必须保证这个文件正确。其次是本地系统与 DNS 服务器之间必须能够进行双向通信，即使用 ping 命令测试能够返回结果。

如果确认系统能与 DNS 服务器进行双向通信，就需要借助域名解析工具判断当前使用的 DNS 服务是否能正常解析域名。在 Linux 系统中，可以使用的域名解析工具有 dig 和 nslookup 两个。本小节将简单介绍如何使用这两个工具判断 DNS 服务器的工作状态。

1．dig工具

（1）dig 命令没有常用的选项，直接将要查询的域名作为其参数即可。例如：

```
#使用 dig 工具查询域名 www.baidu.com 对应的 IP 地址
#默认情况下 dig 命令将使用系统设定的 DNS 服务器解析
# dig www.baidu.com
#dig 命令的查询过程
; <<>> DiG 9.3.4-P1 <<>> www.baidu.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16275
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 4, ADDITIONAL: 0

;; QUESTION SECTION:
;www.baidu.com.                 IN      A

#解析结果
;; ANSWER SECTION:
www.baidu.com.          668     IN      CNAME   www.a.shifen.com.
www.a.shifen.com.       116     IN      A       119.75.218.45
www.a.shifen.com.       116     IN      A       119.75.217.56

#baidu.com 的权威 DNS 服务器列表
;; AUTHORITY SECTION:
baidu.com.              4072    IN      NS      dns.baidu.com.
baidu.com.              4072    IN      NS      ns2.baidu.com.
baidu.com.              4072    IN      NS      ns4.baidu.com.
baidu.com.              4072    IN      NS      ns3.baidu.com.

#统计信息和使用的 DNS 服务器地址
;; Query time: 6 msec
;; SERVER: 192.168.111.215#53(192.168.111.215)
;; WHEN: Mon Nov  1 23:14:00 2010
;; MSG SIZE  rcvd: 162
```

上面的命令使用系统设置的 DNS 服务器 192.168.111.215 进行解析。命令返回了整个解析的详细过程，并且返回了查询后的结果：与 www.baidu.com 对应的 IP 地址是 119.75.218.45 和 119.75.217.56。

（2）许多时候可能不知道哪个 DNS 服务器可用，如果将每个 DNS 域名服务器 IP 地址

都写入文件/etc/resolv.conf 并一一测试，会浪费许多时间。这时可以指定 dig 命令使用的 DNS 服务器地址。

例如用指定域名服务器解析网易域名对应的 IP 地址：

```
#使用@从指定 DNS 服务器 61.139.2.69 处查询
# dig @61.139.2.69 www.163.com

; <<>> DiG 9.3.4-P1 <<>> @61.139.2.69 www.163.com
; (1 server found)
;; global options:  printcmd
;; Got answer:
……
```

通过这种方式，管理员可以从众多的 DNS 服务器中筛选可用的域名服务器，并将其 IP 地址写入配置文件。

2. nslookup工具

除了 dig 工具以外，Linux 系统还提供了另外一个域名解析工具 nslookup。与 dig 不同的是，nslookup 命令显示非常简单，并且还使用了一个可交互式的环境供用户使用。

（1）nslookup 工具也可以像 dig 命令那样，直接将要查询的域名作为参数。例如使用 nslookup 命令查询域名对应的 IP 地址：

```
#使用 nslookup 命令查询域名 www.baidu.com 对应的 IP 地址
# nslookup www.baidu.com
#以下显示的为 nslookup 使用的 DNS 服务器地址
;; Truncated, retrying in TCP mode.
Server:         192.168.111.215
Address:        192.168.111.215#53

#以下为查询到的结果
Non-authoritative answer:
www.baidu.com   canonical name = www.a.shifen.com.
Name:   www.a.shifen.com
Address: 119.75.217.56
Name:   www.a.shifen.com
Address: 119.75.218.45
```

（2）如果需要更换域名服务器查询，就需要进入 nslookup 的交互模式，并使用 server 命令切换 DNS 服务器。

例如要切换 DNS 服务器并查询域名：

```
#直接使用 nslookup 命令进入交互模式
# nslookup
#使用 server 命令切换到 IP 地址为 61.139.2.69 的 DNS 服务器
> server 61.139.2.69
Default server: 61.139.2.69
Address: 61.139.2.69#53
#切换 DNS 服务器后查询域名
> www.baidu.com
#注意以下查询服务器的变化
```

```
Server:         61.139.2.69
Address:        61.139.2.69#53

#以下为查询到的结果
Non-authoritative answer:
www.baidu.com   canonical name = www.a.shifen.com.
Name:   www.a.shifen.com
Address: 119.75.218.45
Name:   www.a.shifen.com
Address: 119.75.217.56
#退出交互模式可以使用 exit 命令
> exit
```

从上面的示例命令输出中可以看出，使用 server 命令切换查询的 DNS 服务器后，nslookup 命令查询的 DNS 服务器发生了变化。

注意：系统从 DNS 服务器解析域名时，使用的目标端口为 53，因此设置防火墙时需要保持这个端口可用。

10.5.5　排除网络故障

对于许多初学者而言，网络非常复杂，一旦系统出现网络故障，许多初学者都会束手无策。对此建议初学者在学习时，应该首先了解系统如何运用网络进行通信、整个通信过程系统采取的步骤等内容。本小节将从系统的角度出发，讲解系统与网络通信的过程、排除故障的思路等内容。

1．计算机与网络通信的过程

为了理解网络故障产生的原因，必须要了解计算机与网络通信的过程，此处将从系统角度出发，简单讲解其通信的过程。对数据发送进行简单分步，可以分为域名转换、数据打包、数据发送 3 个步骤。

（1）域名转换

计算机在对发送的数据进行封装之前，会对目的地址进行检查，如果目的地址是一个域名而非一个 IP 地址，就会对域名进行转换。域名转换时，系统会首先检查系统 hosts 文件中是否含有与域名匹配的记录，如果有匹配的记录则直接使用 hosts 文件中的 IP 地址，如果没有匹配的内容，则会查询 DNS 服务以获取 IP 地址。

（2）封装数据包

系统获取了目的主机的 IP 地址后，会对数据进行封装，首先是网络层使用 IP 协议进行封装（目前都使用 IP 协议），对数据进行拆分、封装并加上 IP 包头等。接着数据链路层会再一次封装，数据链路层封装会加上数据链路层地址（即 MAC 地址）和校验信息（用于保证数据包在传输过程中不被破坏）等，最后将数据包封装成帧交给物理层。

提示：关于网络层、数据链路层封装及转发等相关内容，可以参考网络相关书籍中的 ISO 七层模型、IP 四层模型等。

在数据链路层对数据包进行封装时，会首先判断目标主机的 IP 地址是否与当前主机的网络地址处于同一网段（网络号相同即称两台主机位于同一网段）。对于同一网段的目标主机，系统会使用 ARP 协议（Address Resolution Protocol，地址解析协议）查询目标主机的 IP 地址对应的数据链路层地址（数据链路层地址就是 MAC 地址），然后使用该地址对数据包进行封装。如果目标主机与当前系统不在同一网段，系统会使用 ARP 协议获取默认网关地数据链路层地址，然后使用默认网关的 MAC 地址封装数据包，将其发送给网关，由网关转发数据包。

（3）数据发送

数据包经过两次封装之后，最终会通过物理层传输出去。按使用的网络类型的不同，需要通过交换机、路由器等网络设备寻址，最终将数据包发送到目标主机。需要注意的是，由于此处仅从系统的角度出发，因此交换机、路由器等设备并未在讨论范围之内。

上面简单介绍了数据包发送的 3 个步骤，在实际情况中，这 3 个步骤远比上面讲述的要复杂，读者可以参考网络方面的相关书籍详细了解。

了解了数据包的产生、封装和发送之后，可以简单地分析这 3 个步骤中可能出现故障的地方。

- 域名转换：hosts 文件被篡改，DNS 服务器故障，其他原因导致的域名解析失败（例如链路出现问题导致无法与 DNS 服务器通信）。
- 封装数据包：系统遭遇 ARP 攻击等无法获取到正确 MAC 地址的情况。
- 数据发送：这是最容易出现问题的阶段（占所有故障的 90%以上）。交换机、路由器等设备故障可能会导致数据包发送失败，线路断开等链路故障也有可能会导致数据包无法传输。

排除故障时，需要从最容易出现故障的地方开始逐一排查，即先检查数据发送过程，然后检查域名转换和封装数据包过程。

2．排除网络故障的思路

当系统与网络连接出现困难时，应该按从易到难的顺序检查可能出问题的环节（即最先检查最容易出现问题的环节）。下面将简单讲解这个循序渐进的过程。

（1）ping 命令测试

当应用程序使用网络出现故障时，首先应该使用 ping 命令测试与目标主机是否能够进行双向通信。如果测试结果正常，就可以判定故障点并未出现在数据发送过程中。很有可能是域名转换或封装数据包的过程出现问题，导致无法通信。

如果双向通信测试结果不正常，而目标主机与当前系统处于同一网段，则可能出现的情况是网络线缆断开、交换机故障等原因，应该检查网络线缆、交换机等。如果目标主机与当前系统没有处于同一网段，应首先检查主机与默认网关的连通性，然后再使用 traceroute 命令判断故障发生在哪个路由器附近。

上面这个检查过程如图 10.7 所示。

（2）检查物理层设备和线缆

统计表明大多数的网络故障都发生在物理层，物理层主要由各种网络线缆、光纤、光纤收发器（一种光信号与电信号的转换设备）、中继器（主要作用是放大器，数字通信中还可以排除干扰）等设备组成。检查物理层设备和线缆有以下建议。

- 如果不能确定是否是由于网络线缆导致的故障，通常可以使用网络线缆测试仪、

光信号测试仪进行测试。也可以使用备用线缆替换现有线缆的方法。

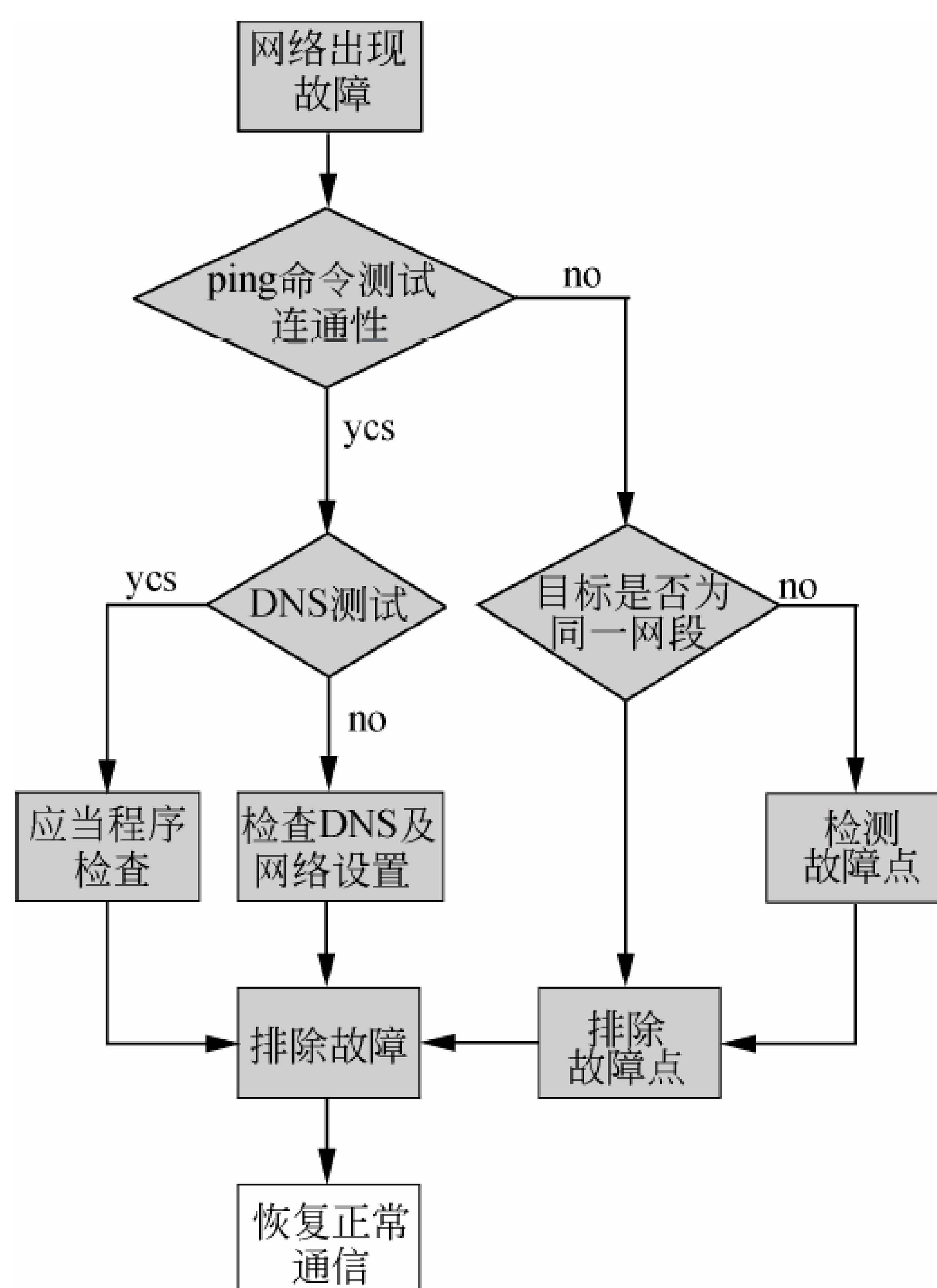


图 10.7　故障检查流程

- ❑ 如果网络中的物理层设备较多，则必须使用排除法对所有设备、网络线缆一一测试排除。
- ❑ 由于与目标主机间可能会有多个节点，这些节点间的任何线缆出现问题都有可能导致无法连接的情况，因此应该使用 ping、traceroute 命令缩小故障范围。

（3）检查 DNS 域名解析

DNS 域名解析检查较为简单，通常只需 5 步即可排除 DNS 域名解析故障：检查 hosts 文件、检查与 DNS 服务器连接是否正常、DNS 服务器是否能正常解析域名、更换 DNS 服务器、防火墙检查。这些检查步骤在域名解析工具中已经介绍过，此处不再赘述。

网络故障通常都十分复杂，而且排除和查找都非常消耗时间，本小节仅仅从系统的角度剖析查找和排除故障的思路。感兴趣的读者可以查阅相关文档，了解网络环境中的故障查找和排除。

10.6　小　　结

- ❑ 10.1 节主要讲解了如何配置网络接口，让系统可以使用网络接口与网络中的其他主机通信。
- ❑ 10.2 节介绍了 Linux 系统中的路由管理，包括查看路由表，添加、删除路由条目等内容。
- ❑ 10.3 节简单介绍了 Linux 系统中的主机名，包括查看、修改主机名等内容。

- 10.4 节讲解了 Linux 系统中的网络负载均衡和网络冗余技术，这是一个非常实用的技术，普遍应用于大中型企业中。
- 10.5 节主要介绍了 Linux 系统中的网络工具、排除网络故障的思路等内容。包括测试连通性命令 ping、网络状态查看命令 netstat 和故障点查找工具 traceroute 等。利用这些工具，管理员可以快速查找网络故障。

总体而言，Linux 操作系统的网络功能十分强大，配置方法十分灵活。本章除了介绍 Linux 系统中的网络配置以外，还介绍了一些较为实用的技术和工具，这些技术和工具在企业中的应用都十分广泛，因此初学者需要注意。

第2篇　文本编辑器

第 11 章　Vi 和 Vim 文本编辑器

与 Windows 系统不同，在 Linux 系统中，服务和系统的设置都保存在文本文件中。要进行某个设置时，只需要使用文本编辑器修改文本文件中的设置，然后让相应的程序读取配置文件即可。由此可以看出文本编辑器在 Linux 系统中的重要性。

Linux 系统中有许多优秀的文本编辑器，其中最出名的是 Vi 和 Vim 编辑器。本章将简单介绍如何使用 Vi 和 Vim 编辑器编辑文本，涉及的主要内容如下。

- 文本编辑器的作用、简单分类，Linux 系统中常见的文本编辑器简介。
- 简单介绍 Vi 编辑器在 Linux 系统中的地位、基本概况。
- 介绍 Vim 编辑器的使用方法、技巧。
- 讲解如何在 Vim 编辑器中快速移动光标。
- 介绍 Vim 编辑器中的窗口操作。
- 讲解 Vim 编辑器中的高级技巧：复制、剪切、粘贴文件，同时编辑多个文本，Visual 模式等。
- 简单介绍如何定制 Vim 编辑器，从崩溃的编辑器中恢复数据等。

11.1　文本编辑器概述

文本编辑器是对纯文本进行编辑操作的一类软件，Windows 系统中的记事本就是一个文本编辑器。与 Word 等字处理软件不同，文本编辑器的对象是普通的文本内容，而非对文档进行复杂的排版、格式处理。本节将简单介绍文本编辑器的概况。

11.1.1　文本编辑器的发展及分类

计算机诞生之初并没有文本编辑器，人们使用穿孔的纸带向计算机发出指令。后来人们发明了打字机，打字机附带了一个行编辑器，使用时需要将一个文本行输入打字机，执行某个命令即可将这一行打印出来。虽然这个行编辑器能将输入结果实时地显示出来，但也存在很多弊端（例如不能修改上一行等）。

当显示器出现之后，人们又编写了可以实时显示的文本编辑器。直到今天，出现了许多功能和界面各异的文本编辑器，供人们使用。

文本编辑器按编辑范围或方式可以分为行编辑器和全屏幕编辑器。

- 行编辑器：行编辑器是一种非常古老的文本编辑器，一次只能编辑文本中的一行。与打字机一样，在行编辑器中，要编辑当前行之前的内容，往往是非常困难的。由于行编辑器使用起来很不方便，现在人们大多使用全屏幕编辑器代替。尽管如

此，行编辑器在自动编辑方面具有较大的优势，在本书的第 5 章中介绍的 sed、awk 都是由行编辑器演化而来的工具。

- 全屏幕编辑器：现在大多数人都使用全屏幕编辑器。与行编辑器不同，全屏幕编辑器可以编辑文本的开头、结尾等任意位置的内容，并且可以对前后文本内容进行反复修改，操作非常方便。全屏幕编辑器中最具代表性的是 Vi 编辑器，十分简便的操作和强大的编辑功能，为其赢得了不少用户。除此之外，Linux 系统中的全屏幕编辑器还有 Vim、Emacs 等。

除按编辑范围分类外，文本编辑器按运行界面不同，可以分为字符界面编辑器和图形界面编辑器。

- 字符界面编辑器：早期的编辑器大多运行在字符界面中，其中最典型的是 UNIX 系统中的 Vi 编辑器、Emacs 等。这些编辑器可能没有华丽的界面，也可能不支持鼠标操作，然而其功能和效率却十分高。现在许多服务器都运行在字符界面下，因此这些字符界面中的编辑器可能十分有用（例如无须频繁地切换到图形界面、在简易的远程终端中使用等）。常见字符界面编辑器有 Vi、Vim、Emacs、Nano 等。
- 图形界面编辑器：当 Linux 运行于图形界面时，用户会更喜欢使用图形界面中的编辑器。图形界面中最典型的文本编辑器是 Gedit，其操作方法与 Winodws 下的记事本类似。它还提供了语法高亮显示功能，使用这些功能可以编写多种程序和脚本。大多数 Linux 发行版都使用 Gedit 作为其图形界面中默认的文本编辑器。除 Gedit 之外，还有 Kate、Kwrite 等。

除了以上两种分类方式以外，还可以按交互方式分类，分为交互式编辑器和非交互式编辑器。在对文本进行编辑时，交互式编辑器需要用户参与整个编辑过程，而非交互式编辑器不需用户参与整个编辑过程。

可能许多读者觉得字符界面中的文本编辑器使用起来非常不方便，但事实上字符界面中的文本编辑器效率十分高，这是图形界面中的文本编辑器不能相比的。但就易用性而言，图形界面下的文本编辑器往往要胜于字符界面中的文本编辑器。

提示： 本书不讨论各种编辑器之间的优劣，但通常建议读者一定要会熟练地使用 Linux 字符界面中的文本编辑器。否则一旦系统发生故障，无法进入图形界面，修复过程将十分麻烦。

11.1.2　Linux 系统中的文本编辑器

在使用和管理 Linux 的过程中，许多时候都需要使用文件编辑器修改配置文件（例如修改自动挂载文件 fstab 等），正因如此，Linux 系统中有许多非常优秀的文本编辑器。本小节将简单介绍几种常见的文本编辑器。

1. Vi编辑器

Vi 编辑器是许多 UNIX、Linux 系统都默认安装的文本编辑器，主要运行在字符界面中。它诞生于 20 世纪 60 年代，虽然在诞生之初 Vi 编辑器的功能十分有限，但经过不断的修改和完善，Vi 编辑器现在已经非常强大。由于 Vi 编辑器十分小巧，在一些小型系统上

或一些特殊情况下，Vi 编辑器可能是唯一能用的文本编辑器。现在能够熟练地使用 Vi 编辑器已经成为 Linux 系统的一项基本技能，因此初学者应该学会使用 Vi 编辑器。

2. Emacs编辑器

Emacs 编辑器于 20 世纪 70 年代诞生于 MIT（麻省理工学院人工智能）实验室。现在 Linux 系统中安装的是 GNU Emacs，它是一个 GNU 项目的产物，因其十分方便程序员编写代码，而大受欢迎，也因此被移植到许多系统中。

Emacs 从严格意义上讲并不应该被称为一个文本编辑器，而应该称其为一个系统。这是因为它不仅仅是一个文本编辑器，利用它还可以收发电子邮件、登录远程主机和玩游戏等。不仅如此，利用 Emacs 甚至还可以煮咖啡！

3. Vim编辑器

Vim 是 Vi 编辑器的增强版，除了能完全兼容 Vi 编辑器之外，还添加了许多新的功能，例如对编写程序代码的支持等。与 Emacs 一样，Vim 也是一个 GNU 项目。在所有编辑器中，Vim 被认为是唯一能与 Emacs 竞争的对手。

4. Nano编辑器

对于许多初学者而言，像 Vi、Vim 和 Emacs 这种功能强大而操作又十分复杂的编辑器，可能学习起来并不容易，这时可以尝试使用 Nano。Nano 是一个十分小巧、简单的文本编辑器。与其他功能文本编辑器相比，Nano 没有华丽的界面，也没有复杂的功能和操作，仅有一个简单的文本编辑功能，十分适合没有特殊需要的初学者使用。

除了以上这些编辑器以外，还有一些其他的比较常见的文本编辑器，例如 Nedit、Gedit 等。对于普通用户而言，不必过于追求精通许多编辑器，能熟练使用其中 1 至 2 种即可。

11.2　认识 Vi 和 Vim 编辑器

Vi 编辑器是 UNIX 和 Linux 系统中十分常见的文本编辑器，几乎所有的发行版都预装了 Vi 编辑器。从诞生至今虽然已经接近半个世纪，然而其小巧的体积和强大的功能，使其到现在为止仍有许多忠实的用户。本节将简单讲解 Vi 和 Vim 编辑器的入门知识。

说明：由于 Vim 可以兼容 Vi，并且在 RHEL5.3 中默认只安装 Vim，因此本书中将采用 Vim 进行讲解，读者可以自行安装 Vi 并进行测试。

11.2.1　启动 Vim 编辑器

在系统地学习 Vim 编辑器之前，应该首先了解如何启动和退出 Vim 编辑器，以便于进行后续的学习。本小节将讲解如何启动 Vim 编辑器。

【命令格式】

要启动 Vim 编辑器，可以直接使用命令 vim，基本格式如下：

```
vim [option] filename
```

可以直接使用命令 vim 启动编辑器，也可以将文件名作为其参数，启动编辑器直接编辑参数指定的文件。Vim 还有一些选项，但一般很少使用，感兴趣的读者可以阅读相关文档。

【用法示例】

直接使用命令 vim 启动 Vim 编辑器，如图 11.1 所示。



图 11.1　Vim 编辑器运行界面

在 Vim 编辑器的运行界面中，显示了许多帮助信息，用户可以使用这些帮助信息了解 Vim 编辑器的方方面面。在 Vim 编辑器的启动界面中，每一行的左侧存在许多“~”符号，Vim 编辑器使用这种方式表示这一行为空。由此可以看出，当前文本编辑器中不存在任何内容。

【Vi 编辑器别名】

如果使用命令 vi 启动 Vi 编辑器，启动后的界面仍然与图 11.1 所示相同，即启动的是 Vim 而不是 Vi。这是因为在 RHEL5.3 中，命令 vi 其实是 vim 的别名。使用别名启动的 Vim 编辑器将完全兼容 Vi 编辑器，并且使用别名启动的 Vim 编辑器还具有体积小、启动速度快等特性，实际上就是 Vim 编辑器的 Vi 兼容模式。

使用 root 用户可以查看其别名设置文件：

```
#使用 cat 命令查看 vi 的别名设置文件内容
# cat /etc/profile.d/vim.sh
if [ -n "$BASH VERSION" -o -n "$KSH VERSION" -o -n "$ZSH VERSION" ]; then
  [ -x /usr/bin/id ] || return
  [ `/usr/bin/id -u` -le 100 ] && return
  # for bash and zsh, only if no alias is already set
  alias vi >/dev/null 2>&1 || alias vi=vim
fi
```

由于别名及使用 Vim 的帮助信息等原因，使用 vi 命令启动的 Vim 编辑器在部分命令和操作上可能仍然存在少许差异。

提示：由于本章包含了许多具体的操作，建议初学者多进行练习，以达到熟练使用的目的。如果手边没有适合练习的文本文件，可以复制一个配置文件（例如 Samba 服务的配置文件/etc/samba/smb.conf）进行练习。

11.2.2　Vim 编辑器帮助

在 Vim 编辑器的首界面正中，是当前正在使用的 Vim 编辑器版本及一些帮助信息，通过使用 Vim 编辑器自带的帮助信息，可以快速地掌握 Vim 的使用技巧。本小节将简单介绍如何查看 Vim 编辑器的帮助信息。

（1）如果要使用 Vim 的帮助信息，可以在编辑器中输入以下命令：

```
:help
```

使用上述命令时，输入的命令应该显示在编辑器最底部。如果不是，可以按 Esc 键返回到命令模式下（命令模式将在 11.2.4 小节中介绍），再输入以上命令。

使用上述命令之后，编辑器将会显示其帮助主界面，如图 11.2 所示。

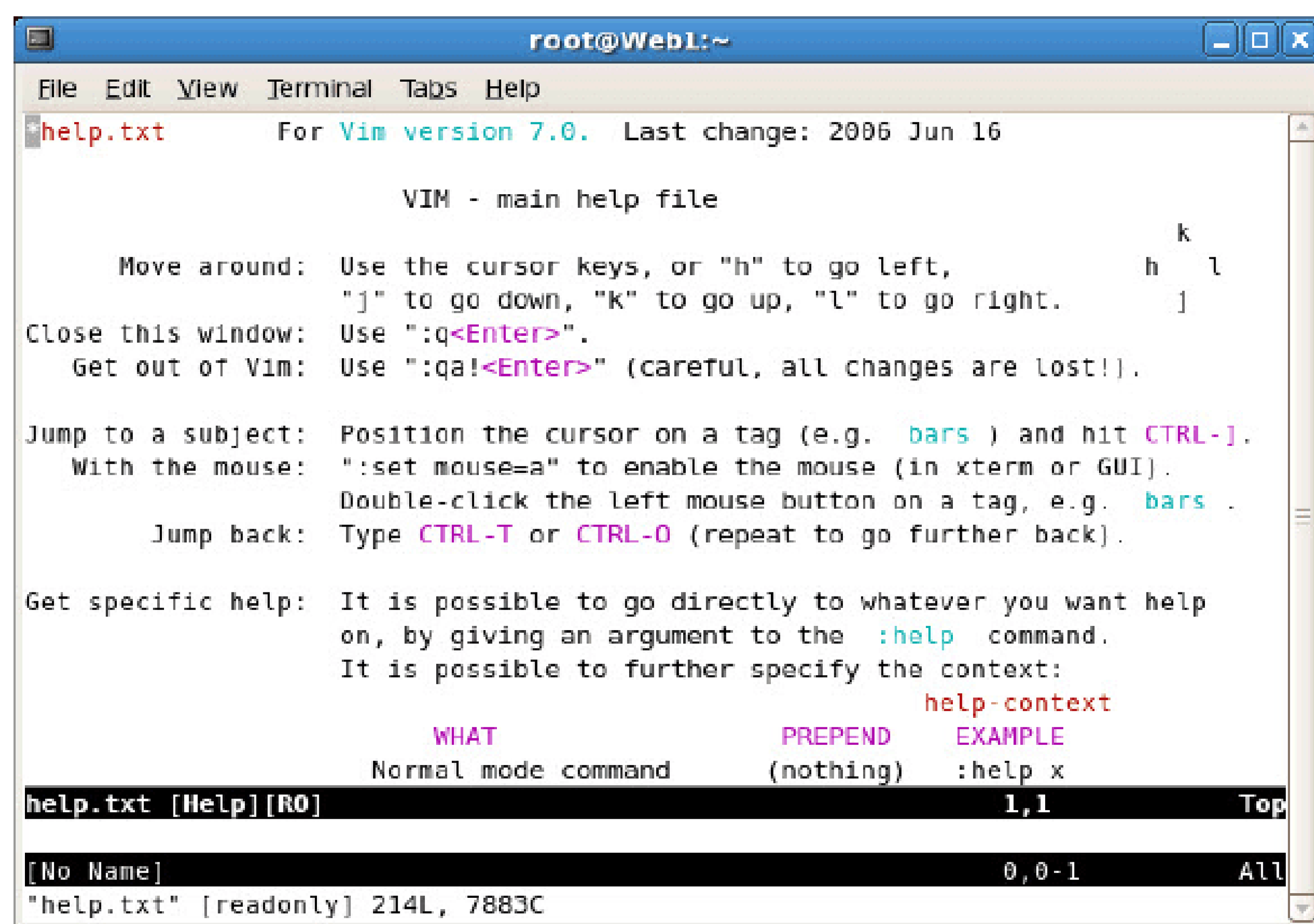


图 11.2　帮助主界面

在帮助主界面中，介绍了如何移动光标、如何退出编辑器、使用鼠标、使用超链接及如何使用其帮助信息等内容，可以通过这些帮助信息快速熟悉 Vim 编辑器的使用方法。

（2）在帮助界面中还使用了一些链接，这些链接将帮助文件放入两个竖线“|”中，形式如下：

```
|usr_01.txt|  About the manuals
|usr_02.txt|  The first steps in Vim
|usr_03.txt|  Moving around
```

其中 usr_01.txt、usr_02.txt 等都是帮助文件名。如果要跳转到链接所在的文件，将光标移动到文件名上使用 Ctrl+]快捷键。阅读完成后，使用快捷键 Ctrl+T 或 Ctrl+O 即可回到帮助主界面。

（3）如果在查看帮助时需要退出帮助，可以使用以下命令：

```
:q
```

（4）除了以上帮助以外，还可以要求 Vim 编辑器显示相关主题的帮助，可以通过在 help 命令之后加入要搜索的帮助主题来实现。例如需要查看关于 Vim 编辑器的信息：

```
:help iccf
```

此时可以查看关于 Vim 编辑器的信息，包括 Vim 的官方网站、使用授权、版本变动等。

（5）如果使用 Vim 编辑器的第 7 版之前，已经学习了其他版本的 Vim 相关操作，要查看当前版本的不同之处，可以使用如下命令：

```
:help version7
```

此命令可以查看当前版本与其他版本之间的差异、当前版本的新功能及其详细使用说明等内容。

11.2.3　退出 Vim 编辑器

对文本编辑结束之后，通常需要退出编辑器。退出编辑器又分为 3 种情况：正常退出、保存退出及强制退出。本小节将简单介绍如何退出 Vim 编辑器。

无论使用何种方法退出编辑器，都必须使编辑器回到命令模式（Vim 编辑器的模式将在下一小节中详细介绍），回到命令模式可以使用 Esc 键。

（1）正常退出的前提条件是：打开的文本文件在内容上没有被改动、新建的文本文件没有添加任何内容、修改的文本文件已经保存等。此时可以使用正常退出命令进行退出，命令如下：

```
#正常退出 Vim 编辑器命令
:q
```

输入以上命令时，命令应该显示在编辑器的最后一行。如果不是，则需要使用 Esc 键重新输入，输入完成后，按 Enter 键即可执行。

（2）如果退出时需要将已经编辑过的文本保存到文件中，可以使用以下命令保存文件并退出：

```
#保存并退出编辑器
:wq
```

（3）如果当前编辑的文本还没有明确指定路径和文件名，或者需要将当前编辑的文本另存，这时可以在保存退出命令后加上文件名：

```
#保存退出时，将文件保存到参数 filename 指定的文件中
:wq filename
```

（4）如果需要强制退出编辑器，可以使用如下命令：

```
#强制退出并不保存
:q!
```

需要注意的是强制退出将会丢失已经编辑的内容，因此在使用强制退出命令之前，应该确保没有保存的内容已经没有任何价值。

11.2.4　Vim 编辑器的模式

Vim 编辑器拥有 3 种基本模式，分别是命令模式（command mode）、插入模式（insert

mode）和末行模式（last line mode）。在不同的模式中，可以执行的任务有所不同。

- 命令模式（command mode）：命令模式中可以使用方向键、编辑键等实现移动当前光标位置、翻页等功能，使用几个简单快捷操作还可以删除单词、行。
- 插入模式（insert mode）：插入模式的主要功能是编辑文本内容。在插入模式中，可以使用方向键、编辑键移动当前光标位置、从键盘输入新的内容，也可以更改文本的内容等。
- 末行模式（last line mode）：末行模式可以输入一些命令，这些命令的功能可以是存储文件、读取文件和退出编辑器等。

Vim 启动后会直接进入命令模式，在命令模式中输入命令的前缀（这些前缀可能是“:”、“/”等），就可以进入末行模式。命令执行完成将直接退出末行模式，并返回到命令模式。由于末行模式通常只在输入命令时存在，因此人们通常将其归入命令模式。

要进入插入模式，可以在命令模式下按 i 键（也包括其他一些快捷键），编辑器最后一行将会显示“-- INSERT --”表示此时正处于插入模式，插入模式如图 11.3 所示。

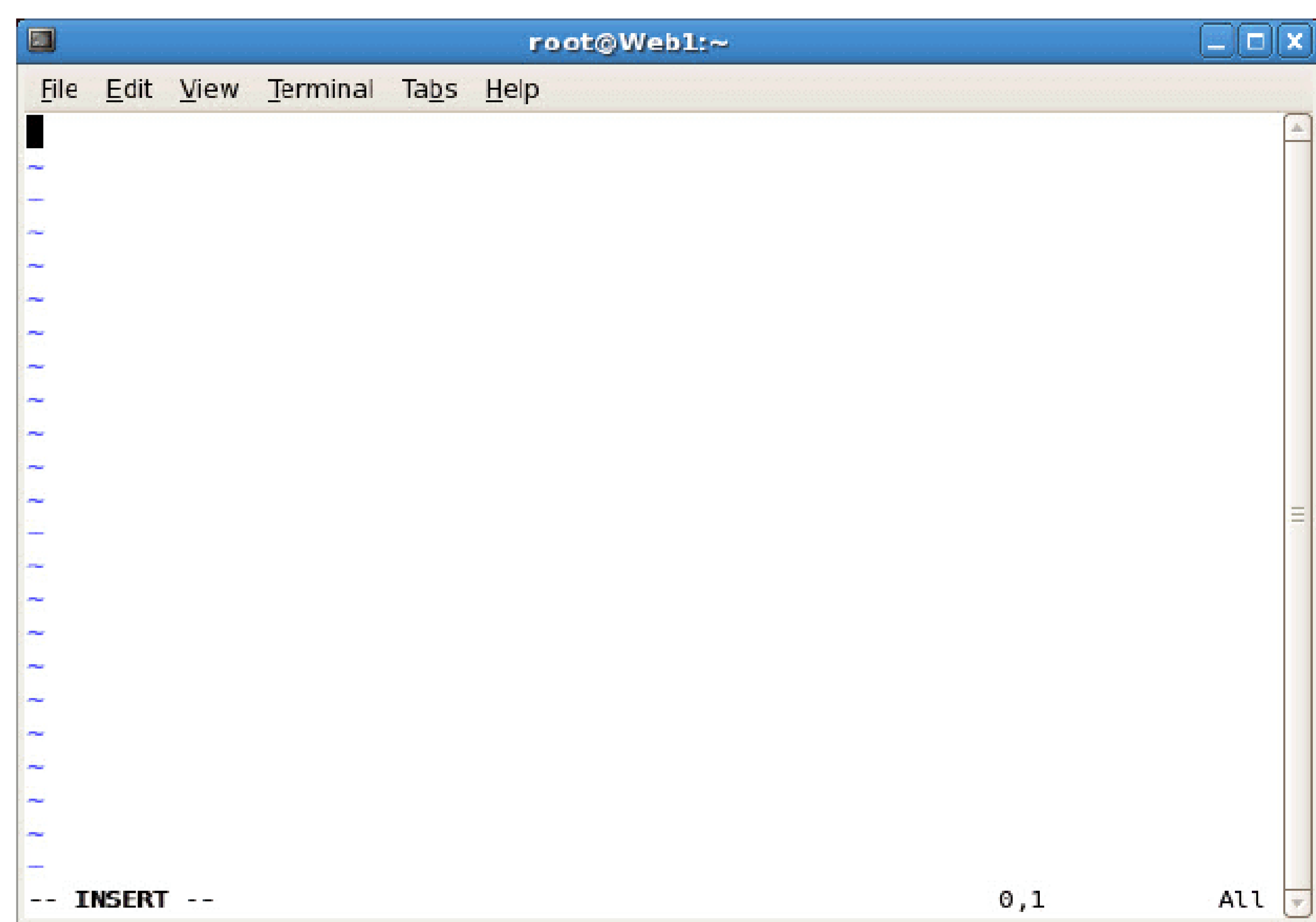


图 11.3　Vim 的插入模式

在插入模式下可以像使用其他文本编辑器一样编辑和修改文本的内容，编辑完成后可以按 Esc 键退出并返回命令模式。

11.2.5　Vim 编辑器的工作界面

使用 Vim 编辑器之前，应该了解 Vim 编辑器界面的组成。本小节将简单介绍 Vim 编辑器的工作界面。

使用 Vim 编辑器打开文本，其界面如图 11.4 所示。

从该界面可以看出，Vim 编辑器使用不同的颜色标识文件中不同类型的文本。例如在配置文件中使用蓝色表示注释、绿色表示选项、红色表示固定值等。

在 Vim 的工作界面中，大部分界面用于显示文本的内容。工作界面最后一行的最左端，一般用于显示状态、信息等内容。例如编辑器进入插入模式时，显示为“-- INSERT --”。最后一行的最右端一般用于显示当前光标所在位置、当前页面占总页面的百分比。例如图 11.4 中，最后一行最右端的“27,1”表示当前光标处于第 27 行的第 1 个字符处，而“9%”

表示当前页面的位置。除此之外，如果编辑器发生错误，也会在此处使用红底文字显示出来。

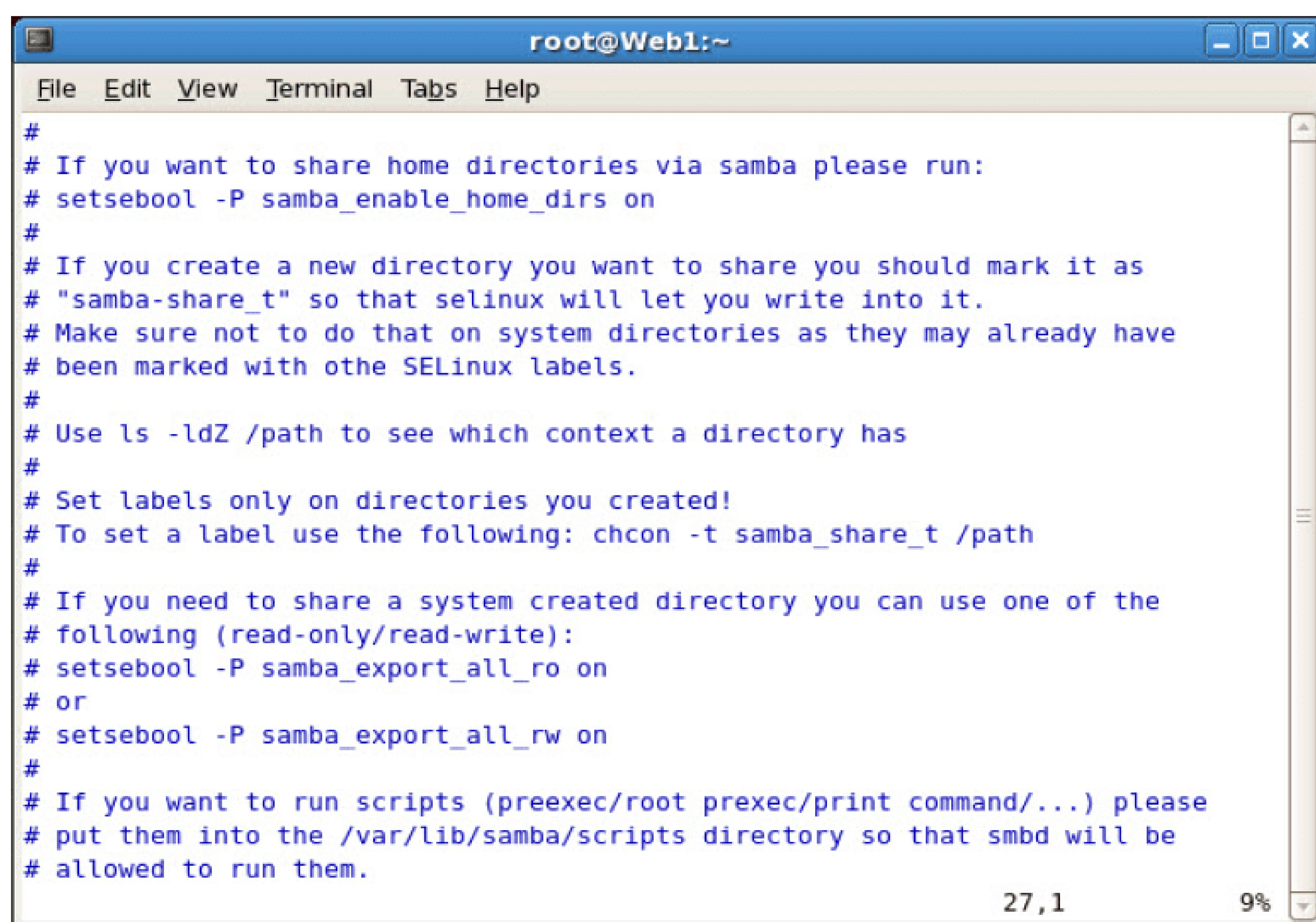


图 11.4　使用 Vim 打开文本界面

11.3　向 Vim 编辑器迈出第 1 步

了解了 Vim 编辑器的启动、退出和工作界面之后，就可以开始使用 Vim 编辑器编辑文件了。本节将简单介绍 Vim 编辑器的基本使用技巧。

11.3.1　读取文件

有时需要从另一个文件读取文本并进行编辑。如果要打开新文件并进行编辑，可以使用命令 e（这个命令类似于图形界面编辑器中的文件→打开）。如果要读取另一个文件的内容，并追加到当前文件的结尾，可以使用命令 r。

（1）例如要打开新文件 a2，使用如下命令：

```
:e /root/a2
```

使用以上命令时，应该保证编辑器中的内容已经保存。如果没有保存，使用 e 命令将会提示错误，如图 11.5 所示。

此时只需要保存即可打开文件 a2。

（2）如果不需要保存当前文件，强制打开新文件，可以使用如下命令：

```
:e! /root/a2
```

（3）要从另一个文件读取文本，并添加到当前文本的最后，可以使用命令 r。例如读取文件 a2 的内容，并追加到当前编辑文本的最后，使用如下命令：

```
:r /root/a2
```

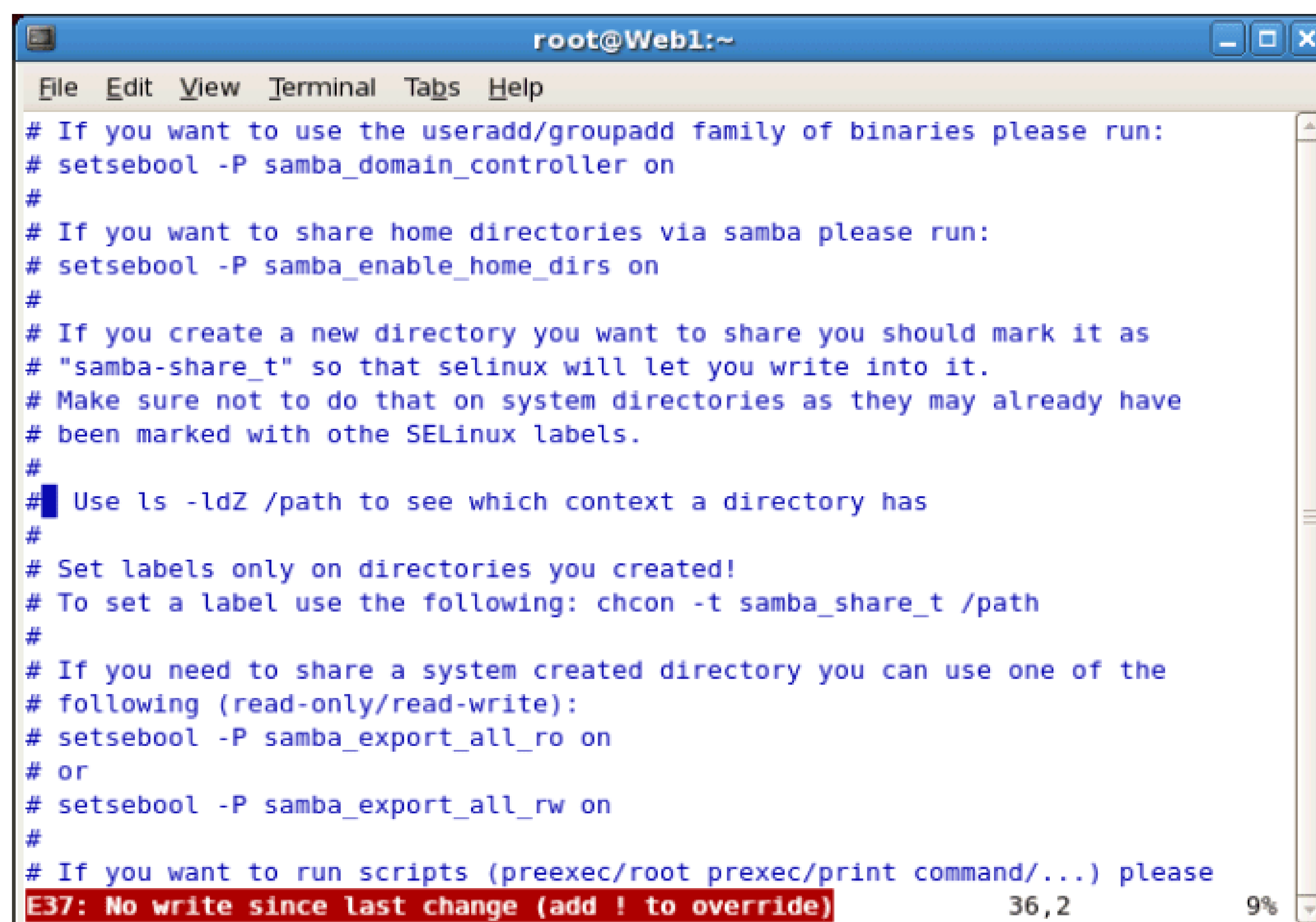


图 11.5　Vim 编辑器提示文本未保存

11.3.2　保存文件

读取和保存文件是一个文本编辑器的基本操作之一。在 Vim 编辑器中，读取和保存文件都使用命令来完成。本小节将简单讲解如何使用命令读取和写入文本。

保存文件命令w

保存文件需要在命令模式中使用命令 w。

（1）例如要保存当前已经编辑的文本：

```
:w
```

如果保存成功，编辑器最后一行会显示提示信息：

```
"a1" 4L, 8C written
```

上面这个提示信息表示，保存的文件名为“a1”（即当前编辑的文件名）。其后的“4L, 8C”表示文件的行数和大小。

（2）如果当前正在编辑的文件还没有命名，或者需要将当前文件另存到另一位置，可以在命令 w 后面加上路径和名称。

例如要将当前文件存放到目录/file 中，并命名为 a2，使用如下命令：

```
:w /root/a2
```

表示路径时，也可以使用相对路径。例如./a2 表示保存到当前工作目录中，并且文件名为 a2。

11.3.3　进入插入模式并插入文本

使用 Vim 编辑器新建一个文件 a3：

```
# vim a3
```

此时 Vim 编辑器将打开一个新的文本，并等待用户输入内容。

（1）要输入内容，还需要进入插入模式。进入插入模式除了使用快捷键 i 之外，还可以使用以下快捷键。

- a：进入插入模式并将光标放在当前光标之后。
- A：进入插入模式并将光标放在当前光标所在的行尾。
- o：在当前光标所在行的后面插入一个新行，进入插入模式并将光标放置在新行。
- O：在当前光标所在行的前面插入一个新行，进入插入模式并将光标放置在新行。

除了以上这些进入插入模式的方法以外，还有一些不常用的快捷键，此处不再一一介绍。

（2）按 i 键，进入 Vim 编辑器的插入模式，就可以自由输入文本了，如图 11.6 所示。

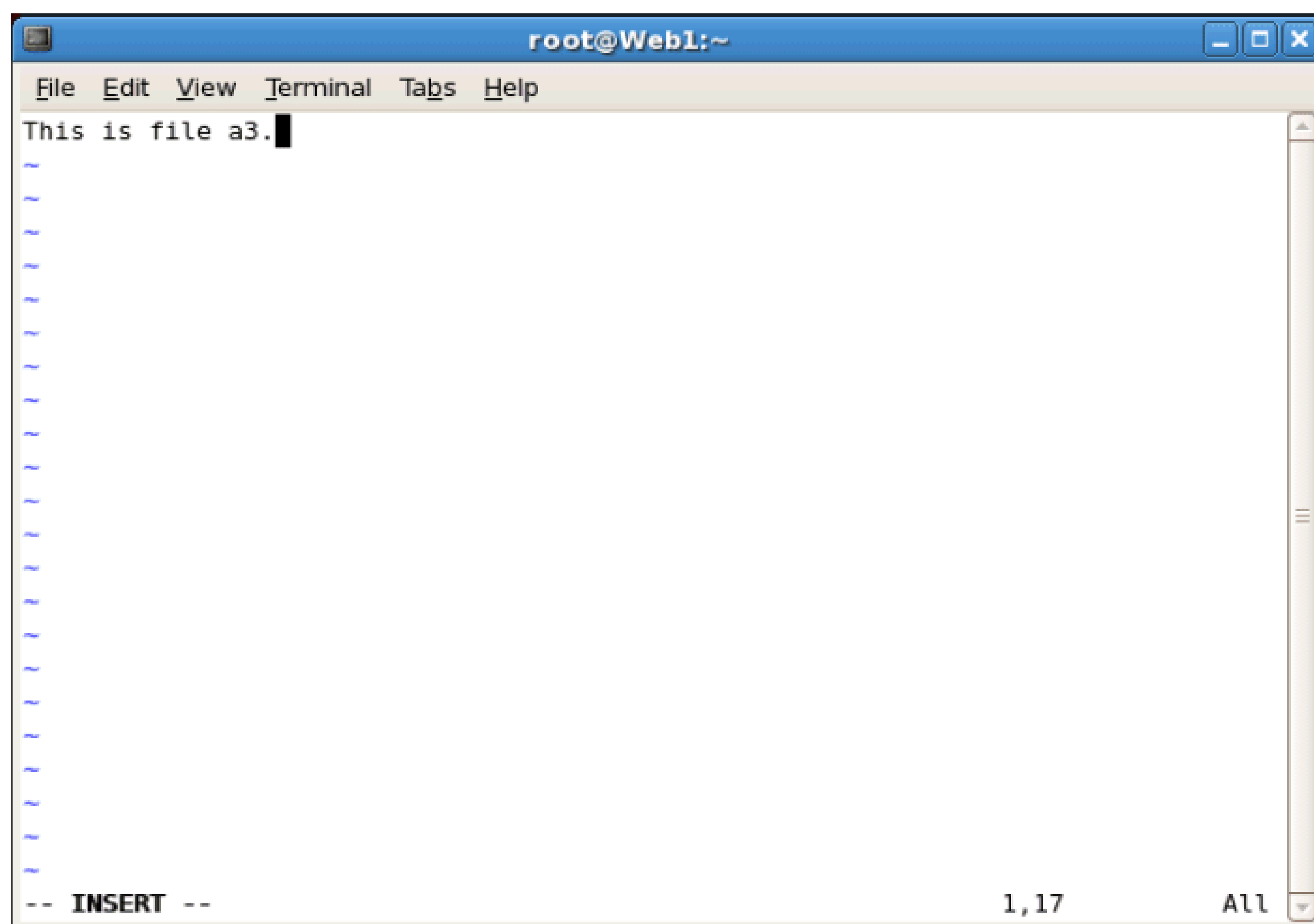


图 11.6　使用 Vim 编辑器编辑文件 a3

当用户输入完成后，可以按 Esc 键进入命令模式，并使用前面介绍的保存命令保存文件 a3。

11.3.4　移动光标

编辑文本内容时，可能需要移动光标反复修改文本内容。在 Vim 编辑器中，无论是在命令模式还是在插入模式中，都可以使用键盘上的方向键移动光标。

除此之外，还可以在命令模式中使用以下快捷键移动光标。

- k：向上移动光标。
- j：向下移动光标。
- h：向左移动光标。
- l：向右移动光标。

当用户将双手放在键盘上时，使用上面这几个快捷键可以非常方便地移动光标（这是因为右手的食指、中指、无名指正好放在 j、k 和 l 键上）。如果觉得这几个快捷不方便记忆，不妨试试记忆快捷键示意图，如图 11.7 所示。

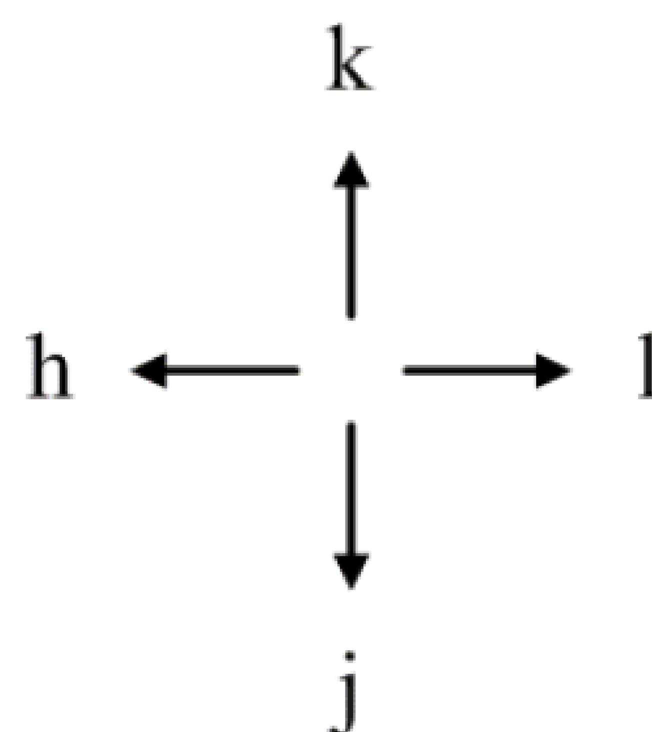


图 11.7　移动光标快捷键示意图

记忆快捷键示意图时，还需要多加练习才能更快、更熟练地使用。

11.3.5　删除文本

许多时候需要删除文本内容，要删除的可能是一个或几个字符，也可能是一行或者几行。本小节将简单介绍如何在 Vim 编辑器中删除文本。

（1）删除单个字符

在插入模式中，可以使用 Back Space 退格键和 Delete 键删除当前光标的前一个字符或当前字符。除此之外，还可以在命令模式中使用快捷键 x 删除单个字符。

使用快捷键 x 时，Vim 会删除当前光标处的字符。如果要删除多个字符，可以连续按下多次 x。

（2）删除多个字符

在使用 Vim 编辑器的过程中，经常需要删除多个字符。为此在命令模式中，Vim 还提供了几个功能更强大的删除快捷键。

- ❑ dd：删除当前光标所在行（dd 是一个较常用的快捷键）。
- ❑ dw：删除当前光标处的单词，包括词尾空格。
- ❑ de：删除当前光标处的单词。
- ❑ d^：删除当前光标到行首的所有字符。
- ❑ d$：删除当前光标到行尾的所有字符。
- ❑ J：将当前光标所在行和下一行合并（相当于删除行尾的换行符）。

在上面的几个快捷键中，使用 dd 删除行时，还可以使用 *n*dd 这样的形式，一次删除 *n* 行。

11.3.6　撤销和恢复

如果执行了某个错误的编辑操作，在图形界面中的编辑器中，可以使用快捷键 Ctrl+Z 撤销已经执行的操作、Ctrl+Y 恢复撤销的操作（有时也称为重做）。在 Vim 的命令模式中，也可以使用以下快捷键执行撤销操作。

- ❑ u：撤销上一步操作，可以多次使用。
- ❑ Ctrl+r：恢复已经撤销的操作，可多次使用。

使用这两个快捷操作可以随意撤销和恢复已经执行了的编辑操作。读者可以输入一行文本并逐一删除，然后练习这两个快捷键的使用。

11.4　快速移动光标

通过前面几节的学习，初学者已经能够使用 Vim 编辑器执行简单的编辑任务了。为了能让初学者提高编辑效率，本节将继续介绍如何在 Vim 编辑器中快速移动光标、定位编辑位置。

11.4.1　按单词移动光标

在编辑文本行时，如果文本行很长，使用方向键移动光标可能会很慢。这时可以按单

词移动光标。

（1）在命令模式中，使用快捷键 b，可以将光标快速移动到当前光标所在单词的前一个单词的首字母。如果要移动到当前光标所在位置的前 3 个单词的首字母处，可以使用 3b。

（2）如果要将光标快速移动到当前光标的后一个单词的首字母，可以在命令模式中使用 w。与快捷键 b 相同，移动多个单词时，可以在快捷键 w 前加上数字。

（3）如果要移动光标至后一个单词的尾字母，可以使用快捷键 e。与前两个快捷键类似，移动多个字母只需要在快捷键前加上相应的数字即可。

（4）与快捷键 e 功能相反的是 ge。使用此快捷键时，Vim 编辑器会快速移动光标至前一个单词的尾字母。

本小节介绍的按单词移动光标的快捷键功能示意图如图 11.8 所示。

使用本小节介绍的几个快捷键，可以在行内快速移动光标。但这几个快捷键记忆起来难度较大，因此推荐在练习中记忆。

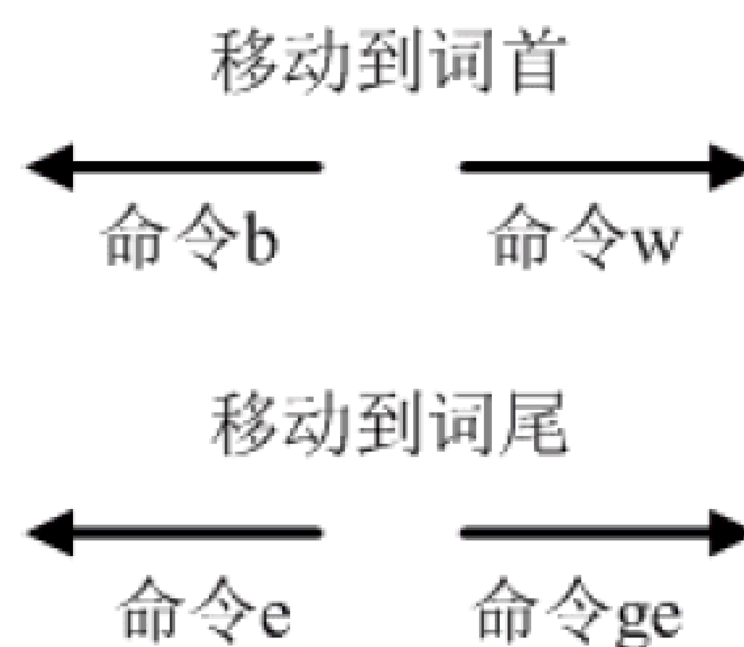


图 11.8　按单词移动光标快捷键功能示意图

11.4.2　快速移动光标至行首和行尾

需要按行快速移动光标时，可以使用键盘上的编辑键 Home，快速将光标移动至当前行的行首。除此之外，也可以在命令模式中使用快捷键“^”（即 Shift+6）或 0（数字 0）。

如果要快速移动光标至当前行的行尾，可以使用编辑键 End。也可以在命令模式中使用快捷键“$”（Shift+4）。与快捷键“^”和 0 不同，快捷键“$”前可以加上数字表示移动的行数。例如使用“1$”表示当前行的行尾，“2$”表示当前行的下一行的行尾。

11.4.3　移动光标至指定的行

调试脚本文件或程序时，可能会提示源代码的某一行有错误（大多数编译器都会以行号的形式提示）。这时就要快速定位到错误行，并执行编辑操作。

（1）为了方便查看行号，Vim 编辑器提供了显示文本行号的功能，但默认情况下这个功能并没有启用，需要手动设置。

要设置显示行号，可以在命令模式中执行：

```
:set number
```

执行上述命令之后，编辑器就会显示行号，如图 11.9 所示。

（2）查看到了行号之后，就可以在命令模式中，使用快速跳转到行快捷键跳转到指定位置。

- gg：跳转到文本的第 1 行，也可以使用 1G。
- G：跳转到文本最后一行。
- *n*G：跳转到第 *n* 行，其中的 *n* 应该是一个正整数。

可以使用以上方法快速跳转到指定行的行首。

（3）如果不需要使用行号，可以在命令模式中使用以下命令取消行号：

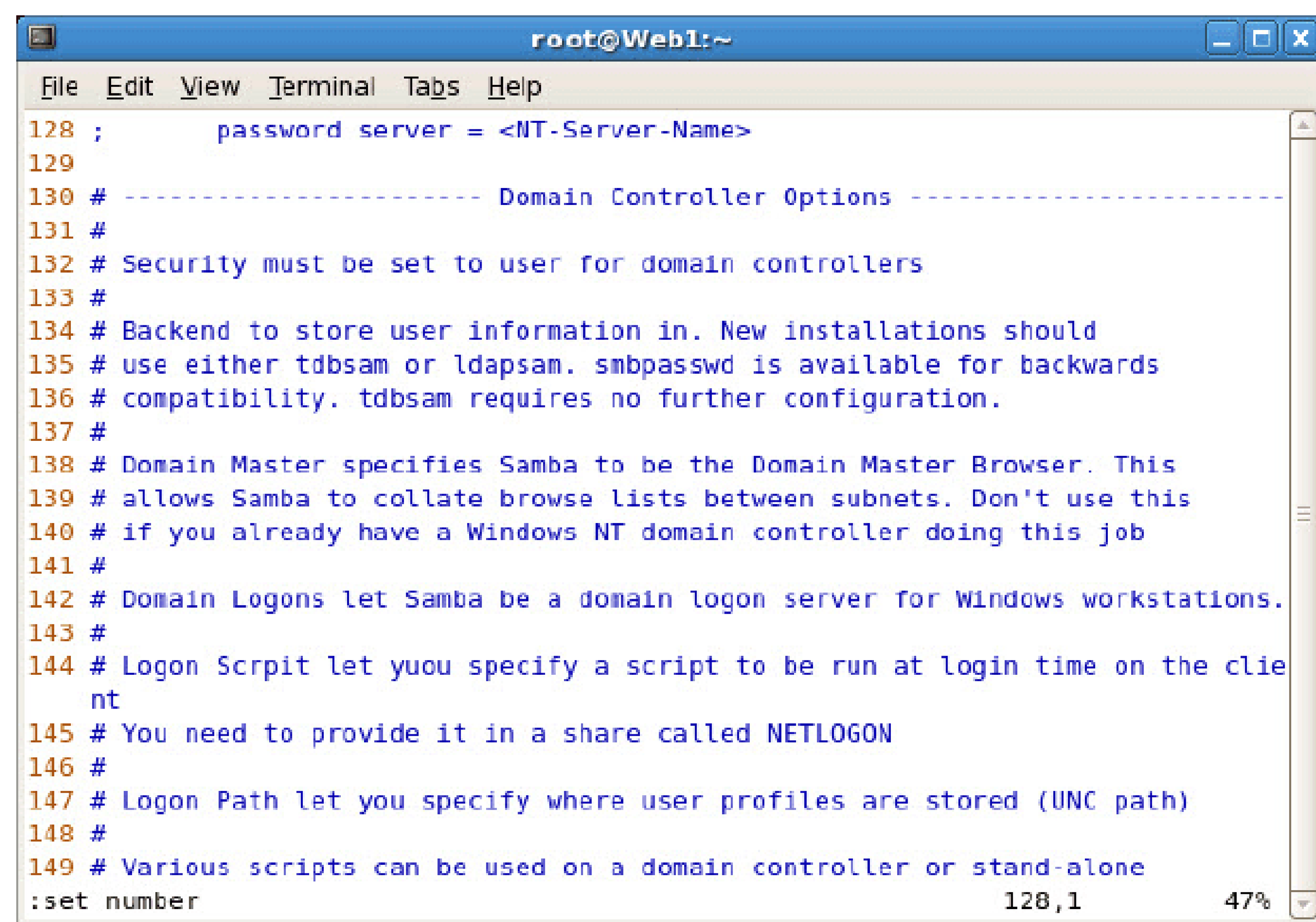


图 11.9　显示行号

```
:set nonumber
```

移动光标到指定行的功能在编程中经常用到，因此使用 Vim 编写程序代码的读者需要掌握此功能的用法。

11.4.4　滚动屏幕

当使用 Vim 编辑器打开一个足够长的文本文件时，Vim 会依次显示文件的第 1 行、第 2 行……直到整个屏幕。如果文本文件超过屏幕长度，图形界面中的文本编辑器通常会在窗口右侧显示一个滚动条，以便使用者滚动屏幕。但 Vim 编辑器没有滚动条，这时可以使用快捷键来完成这个功能。

在命令模式中，常用的滚动屏幕快捷键如下。

- Page Up：向前翻页。
- Page Down：向后翻页。
- Ctrl+B：向前翻页。
- Ctrl+F：向后翻页。
- Ctrl+U：向前翻半页。
- Ctrl+D：向后翻半页。

在插入模式中，不能使用快捷方式翻页，但可以使用编辑功能键 Page Up、Page Down 实现翻页。

11.4.5　使用鼠标移动光标

对于许多人而言，可能更希望在快速定位时能够使用鼠标，这样将会大大提高光标的定位速度。Vim 编辑器也提供了鼠标支持，要使 Vim 编辑器支持鼠标，可以使用以下命令：

```
:set mouse=a
```

此时 Vim 将会弹出鼠标并允许使用鼠标进行光标定位，如图 11.10 所示。

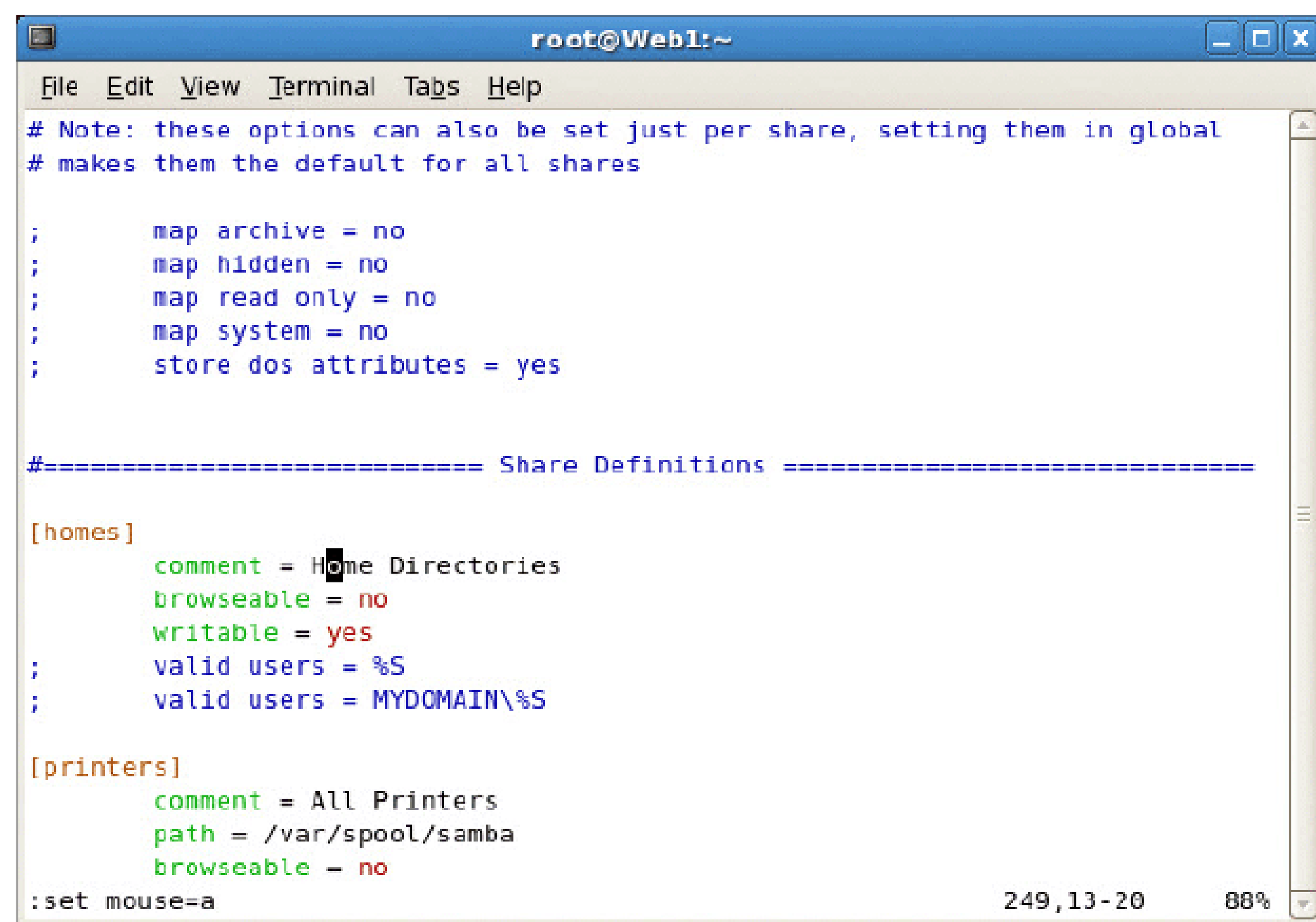


图 11.10　使用鼠标定位

这时就可以像在 Windows 中那样，通过单击鼠标将光标移动到指定的位置。

11.4.6　其他移动光标的技巧

除了前面几个小节中介绍的快速移动光标的技巧之外，Vim 编辑器中还附带了许多移动光标的技巧。本小节将简单介绍几个常见的移动光标的技巧。

1. 快速跳转到编辑位置

许多时候，当使用快捷键跳转到新的位置并编辑之后，还需要再跳转回原来的位置，这时可以使用快速跳转快捷键将光标移动到原来的位置。快速跳转快捷键需要在快捷键模式下使用``，这是两个反引号，在键盘上位于数字键 1 和 Tab 键附近。如果按下多次快速跳转键，Vim 编辑器会在最后两个编辑位置之间跳转。

如果在编辑过程中执行了多次跳转，可以使用以下快速跳转键跳转到更多编辑过的位置。

- Ctrl+I：跳转到前一个编辑位置。
- Ctrl+O：跳转到后一个编辑位置。
- Tab：跳转到前一个编辑位置（与 Ctrl+I 功能相同）。

默认情况下，Vim 将会记录所有的跳转位置，并且用户可以使用记录在多个编辑位置自由跳转。如果要查看 Vim 编辑器记录的跳转位置，可以在命令模式中使用：

```
:jumps
```

此时 Vim 会显示所有跳转记录，包括文件的名称、跳转位置的文本、行号等内容。

注意：虽然 Vim 编辑器保存了许多不同文件的跳转点，但在使用跳转快捷键时，并不会因此跳转到其他文本文件中。

2. 搜索字符并移动光标

搜索字符并移动光标功能与图形界面中的查找不同。搜索字符移动光标时，只能使用

单个字符，并且搜索字符移动的范围为光标所在行（查找功能通常是全文查找）。

（1）搜索字符命令为 fs，其中 f 为搜索命令（可速记为 find），s 为需要搜索的单字符。这是一个非常有趣并且有用的命令。它会在当前光标之后查找单字符 s，并将光标移动到单字符 s 之上。如果当前光标至行尾的范围内没有找到单字符 s，则不移动光标。

与前面介绍的某些命令一样，搜索字符命令也可以与数字一起配合使用。例如 2fn 表示查找当前位置之后的第 2 个字符 n 并移动光标，4fs 表示查找当前位置之后第 4 个字符 s 并移动光标。命令 fs 的功能示意图如图 11.11 所示。

（2）与搜索字符命令 fs 功能相反的是 Fs。它的功能是在当前光标位置之前查找单字符 s，并将光标移动到单字符 s 之上。其功能示意图如图 11.11 所示。

（3）与搜索字符命令功能类似的还有 ts（Ts），这个命令可速记为 to。其功能分别是在当前位置之后（之前）查找单字符 s，并移动光标到单字符 s 之前（之后）。其功能示意图如图 11.11 所示。

在图 11.11 所示的示意图中，需要额外注意使用命令 t 和 T 时，Vim 会将光标移动到搜索到的字符串之前、之后。

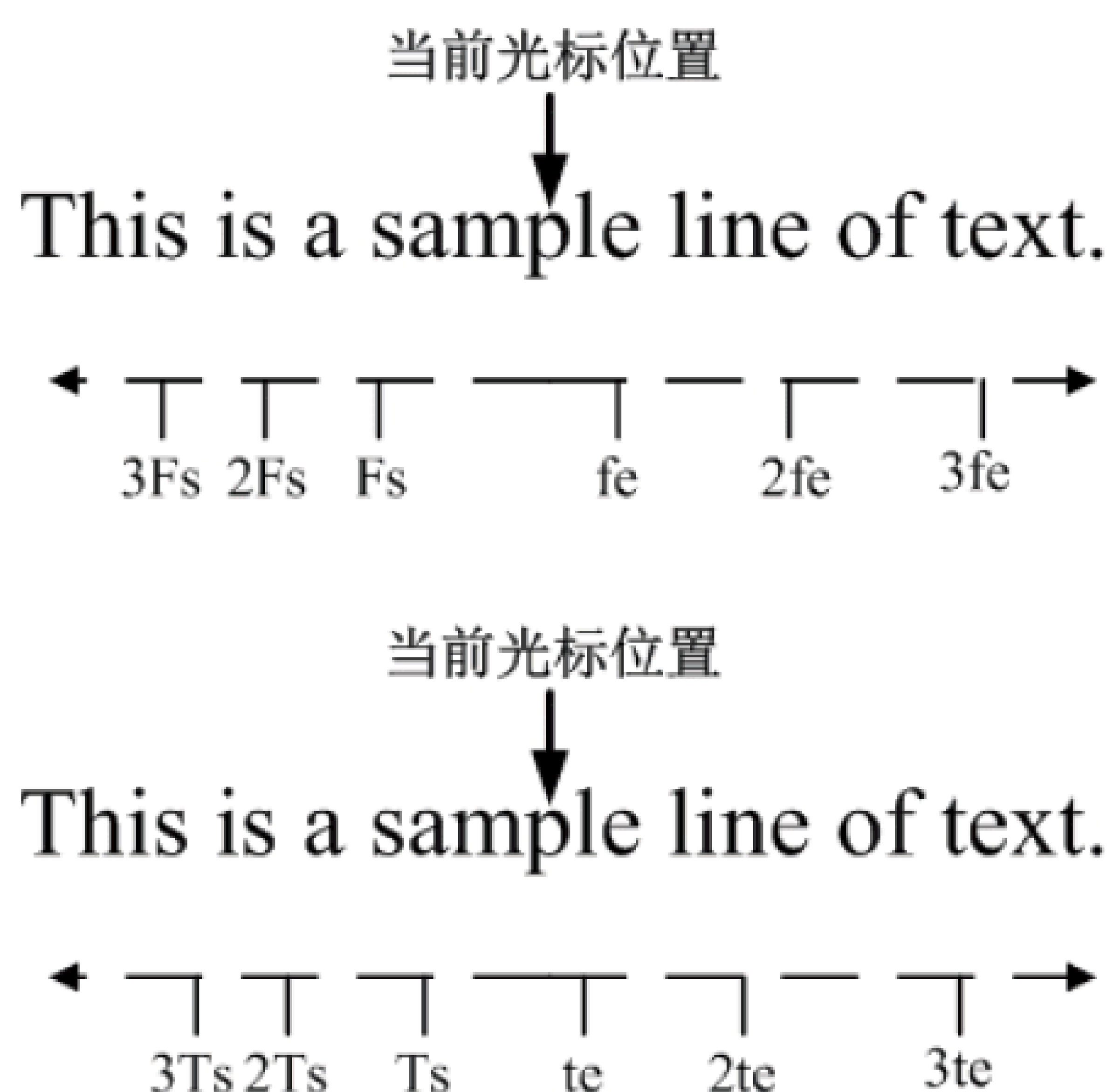


图 11.11　搜索字符移动光标命令功能示意图

技巧： 如果使用 f、F、t、T 命令未将光标移动到指定位置，可以使用快捷键“;”和“,”继续搜索移动光标。

3. 以括号为目标移动光标

在修改程序代码时，程序员经常会被众多嵌套的括号迷惑。特别是当括号较多时，非常容易出现错误。在 Vim 编辑器中，可以以括号作为目标移动光标。

以括号为目标移动光标的快捷键是%（即组合键 Shift+5）。使用方法为先将光标移动到括号上（左、右括号均可），然后使用快捷键，Vim 会自动将光标移动到另一半括号上。快捷键%适用于圆括号“()”、方括号“[]”、大括号。

快捷键%的功能如图 11.12 所示。

在图 11.12 中，快捷键会自动为括号配对。当用户使用快捷键%时，Vim 会自动将光标移动到与当前光标所在括号配对的另一半括号上。

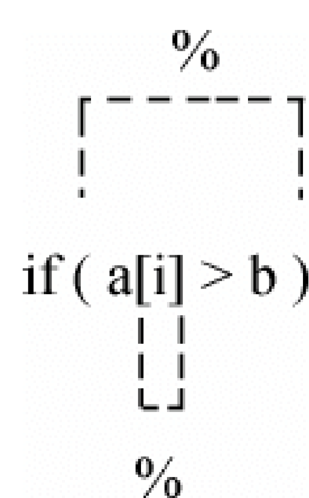


图 11.12　快捷键%的功能示意图

本小节介绍了 Vim 编辑器中几种最常见的快速移动光标的技巧，读者可以自行找一段文本进行练习，以便更快地掌握这些技巧。

11.5　Vim 编辑器的查找和替换功能

许多文本编辑器中都自带了查找、替换功能，Vim 编辑器也不例外。使用这两个功能

不仅可以快速定位编辑位置，还可以批量修改文本。本节将简单介绍 Vim 编辑器中的查找和替换功能。

11.5.1　简单的查找功能

在 Vim 编辑器中，实现简单查找功能的命令是“/”，其格式如下：

```
/pattern
```

在上面的格式中，pattern 为要查找的字符串。

使用命令“/”之后，Vim 编辑器会将光标移动到找到的字符串的第 1 个字符处。例如使用命令/home 查找字符串 home，Vim 编辑器会从当前光标位置开始向下查找，并将光标移动到查找到的字符串的第 1 个字符处，如图 11.13 所示。

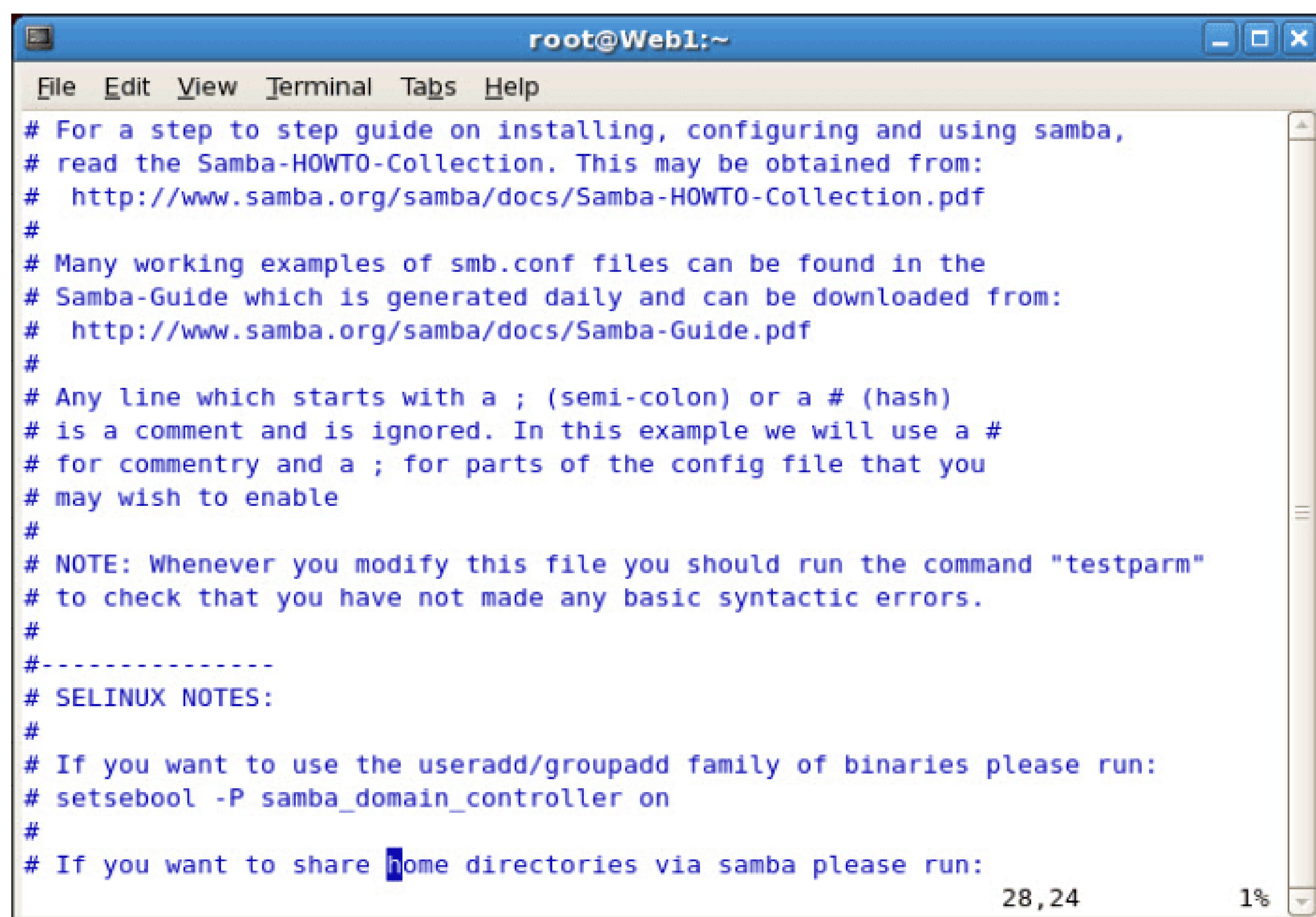


图 11.13　简单查找功能

在图 11.13 中，Vim 编辑器将光标移动到找到的字符串的首字符处。如果当前查找到的字符串不是用户需要查找的字符串，可以使用以下快捷键。

- n：跳转到下一个字符串。
- N：跳转到上一个字符串。

使用上面两个快捷键跳转查找时，如果 Vim 查找到的字符是第一个或最后一个字符串，Vim 将会提示用户，如图 11.14 所示。

在图 11.14 中，Vim 显示错误，并提示用户：当前找到的字符串是最后一个匹配的字符串。

技巧：使用 Vim 编辑器查找字符串时，可以使用命令 set wrapscan 和 set nowrapscan 开启、关闭跳过文件首尾查找功能。

11.5.2　反向查找

在 Vim 编辑器中，除了使用命令“/”从当前位置开始向下查找外，还可以使用命令

“?”执行反向查找，即从当前位置开始向上查找，基本格式如下：

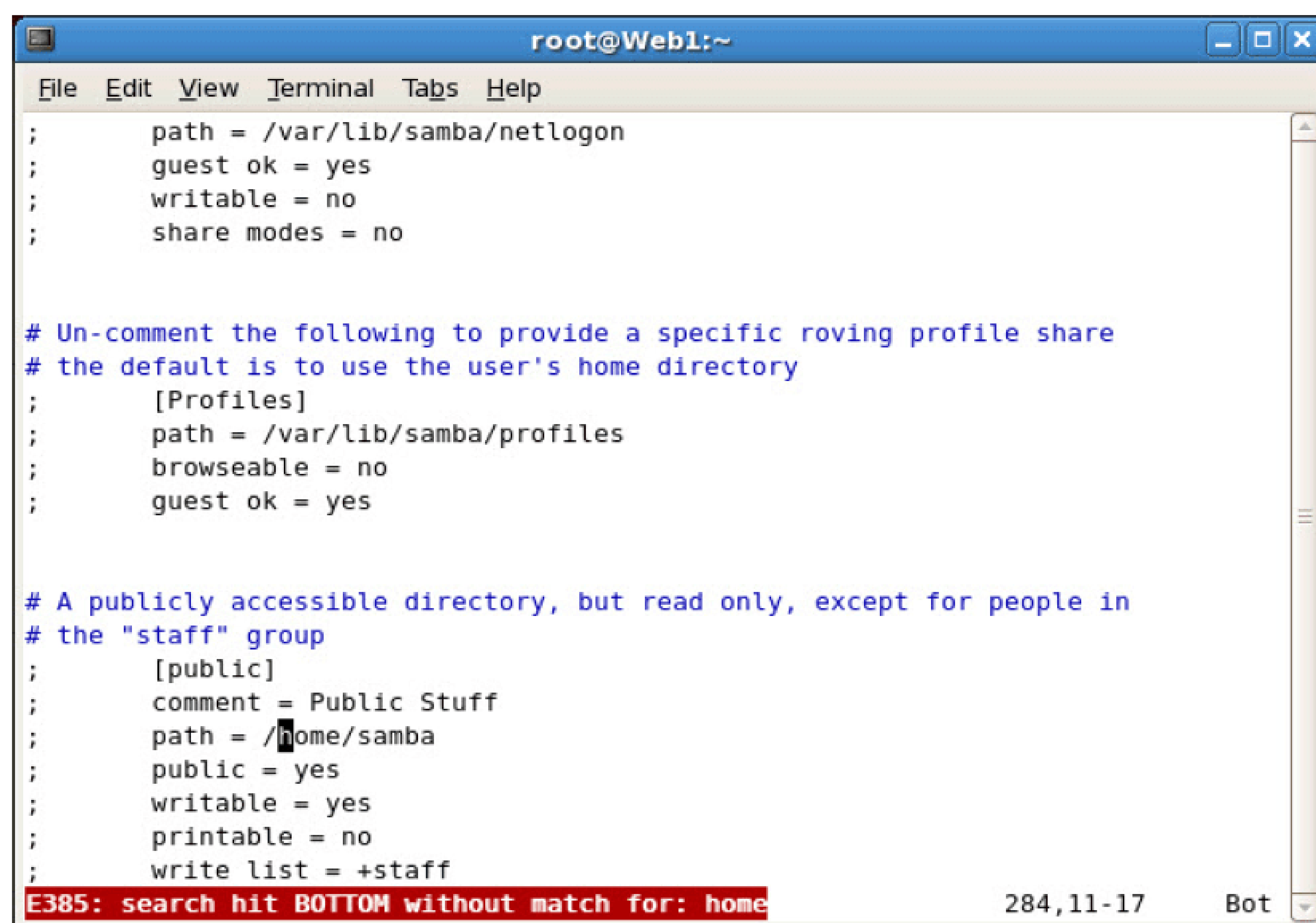


图 11.14　查找时提示用户

```
?pattern
```

反向命令“?”与正向查找命令的使用方法完全相同，主要不同之处在于跳转查找字符串快捷键用法不同。

- n：跳转到上一个字符串。
- N：跳转到下一个字符串。

对比 11.5.1 节介绍的跳转快捷键不难看出，跳转字符串快捷键的功能与正向查找的功能正好相反。读者可以自行练习这两个快捷键的用法。

11.5.3　查找时忽略大小写

使用 Vim 编辑器的查找功能时，Vim 编辑器会“老老实实”地查找与字符串完全匹配的文本，连字母的大小写也不例外。但有时可能并不需要 Vim 这么严谨地工作，这时就需要忽略字符串的大小写。

（1）要让 Vim 编辑器查找时忽略字符串的大小写，可以使用以下命令：

```
:set ignorecase
```

使用以上命令之后，再使用字符串 home 进行查找，Vim 编辑器就能够匹配到以下字符串：

```
Home
home
hOme
HOME
……
```

（2）当然有时开启忽略大小写功能可能会不方便，这时可以使用以下命令关闭忽略大小写功能：

```
:set noignorecase
```

11.5.4　高亮显示查找结果

查找字符串时，让查找到的字符串都高亮显示，以便使用者自由查看，是一个非常好的办法。

（1）要高亮显示所有匹配的结果，可以在 Vim 编辑器的命令模式中执行以下命令：

```
:set hlsearch
```

使用高亮显示搜索结果后，Vim 会将所有匹配项都采用高亮显示，非常方便查看。高亮显示搜索匹配项如图 11.15 所示。

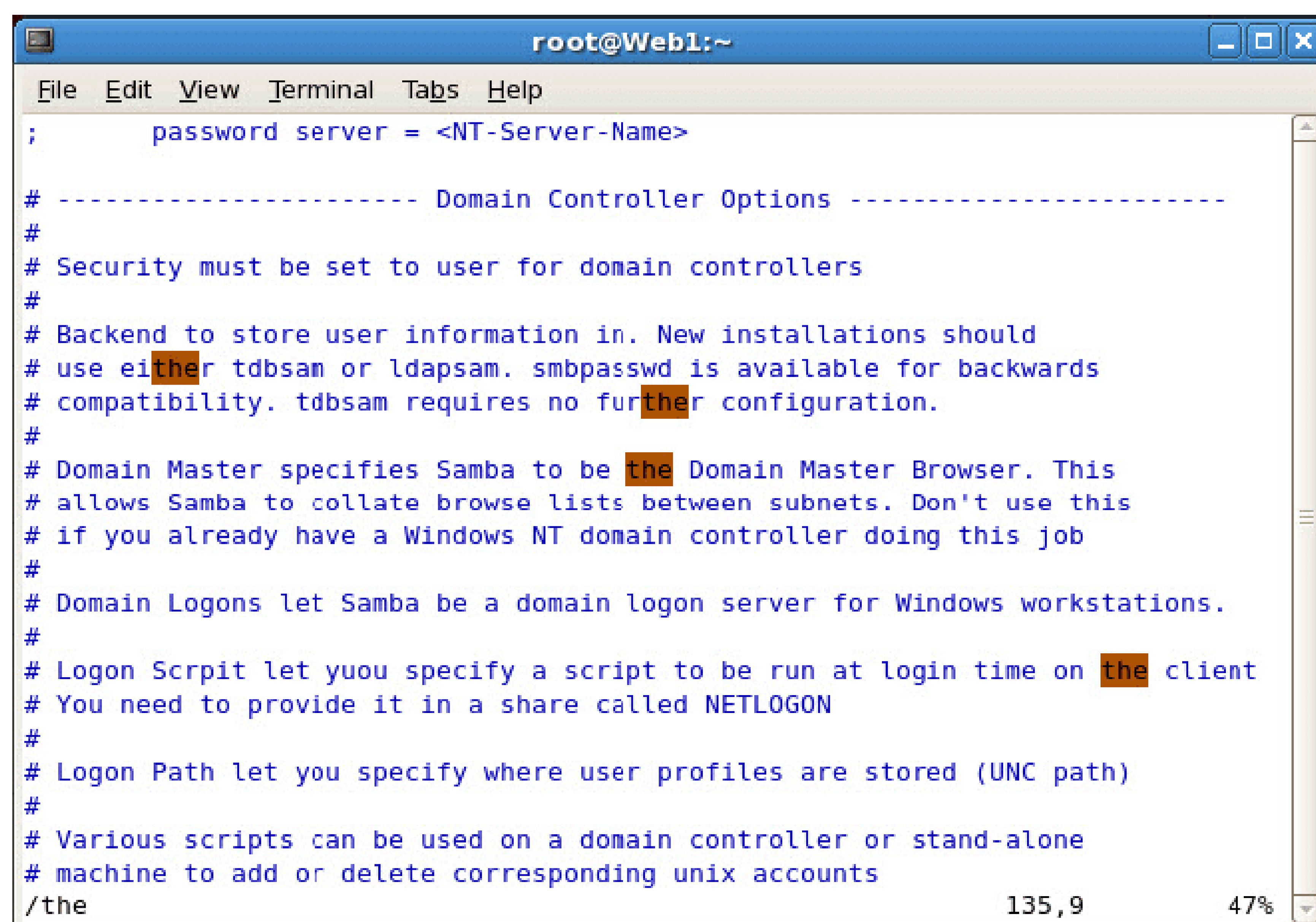


图 11.15　搜索结果高亮显示

（2）在使用搜索结果高亮显示后，可能使用过的搜索项在下一个文本中也是高亮显示的，此时可以使用以下命令关闭搜索结果高亮显示功能：

```
:set nohlsearch
```

使用以上命令关闭了搜索结果高亮显示功能后，如果需要再次使用此功能，必须要手动开启该功能。

11.5.5　增量查找

增量查找是指用户输入需要查找的字符串的同时，Vim 编辑器按用户的输入同步进行查找。默认情况下，Vim 编辑器的增量查找功能处于关闭状态。

（1）要打开 Vim 编辑器的增量查找功能，可以在命令模式中执行以下命令：

```
:set incsearch
```

（2）使用以上命令开启增量查找功能之后，输入查找命令“/ho”，此时 Vim 编辑器的显示内容如图 11.16 所示。

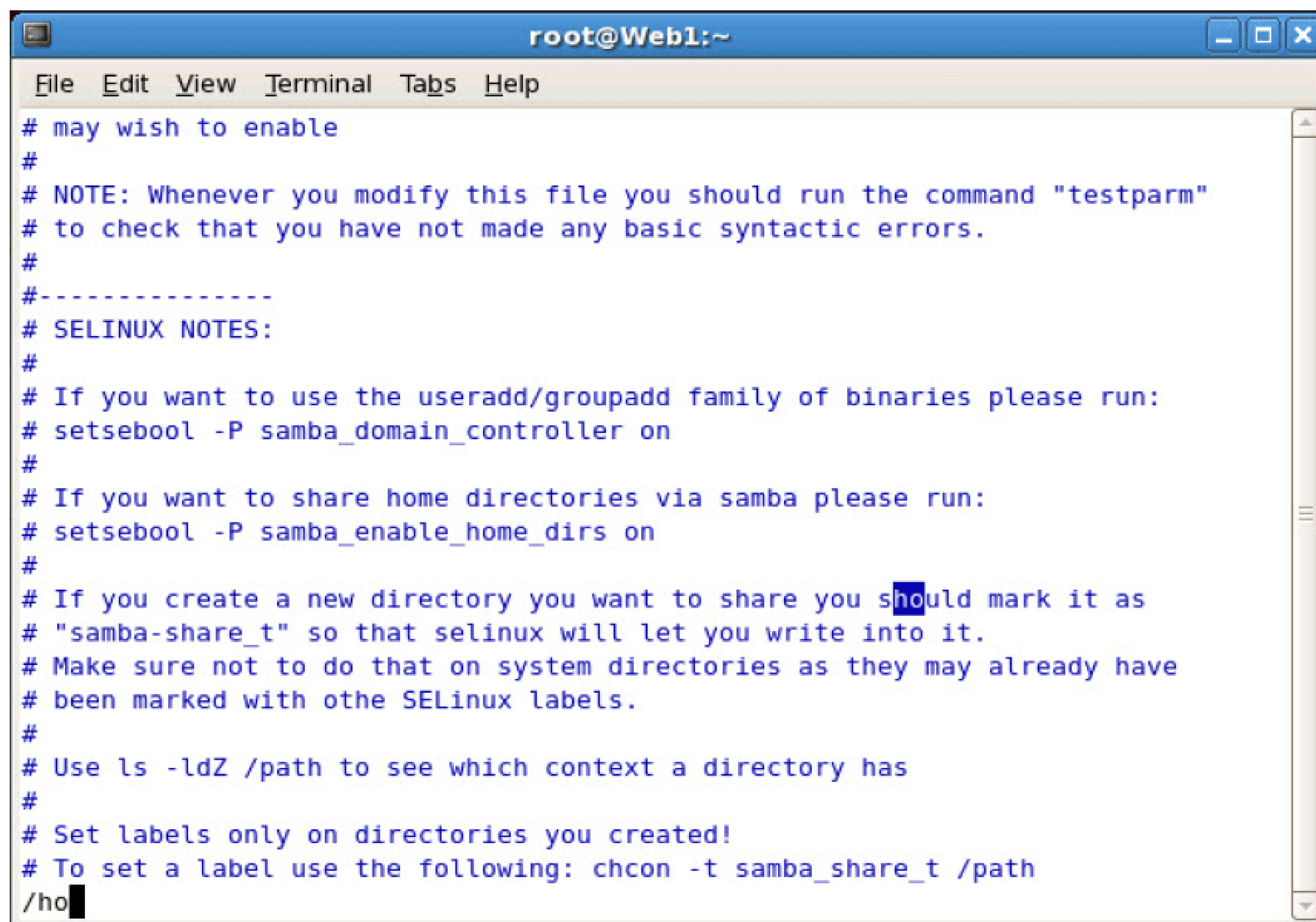


图 11.16　Vim 编辑器的增量查找功能

从图 11.16 可以看出，用户输入查找字符串的同时，Vim 编辑器也开始了查找工作（之前是用户输入命令并按 Enter 键后才开始查找）。

（3）与其他命令相似，要关闭增量查找功能，只需要在命令之前加上 no 即可：

```
:set noincsearch
```

11.5.6　简单的查找替换功能

许多时候需要将文本中的特定字符串做相同的修改，这时使用替换文本功能可以达到事半功倍的效果。本小节将简单介绍 Vim 编辑器的替换文本功能。

在 Vim 编辑器中，替换功能使用的命令是 s，基本格式如下：

```
:s/pattern1/pattern2/
```

在上面这个示例中，pattern1 为要查找的文本，pattern2 为替换后的文本，“/”为分隔符。

（1）虽然本例中使用了“/”作为分隔符，但实际上命令 s 并没有强行规定必须使用“/”作为分隔符。例如可以使用冒号作为分隔符：

```
:s:the:The:
```

在上面这个命令中，s 命令使用冒号作为分隔符。上面的命令的功能是将光标所在行的第 1 个 the 替换为 The。

提示：如果要使用特殊字符作为分隔符，或当替换的字符串中含有特殊字符时，应该使用符号“\”屏蔽其特殊含义。

（2）使用 s 命令进行替换时，Vim 总会替换光标所在行第 1 个匹配到的字符串。如果当前行没有找到需要替换的字符串，Vim 将提示用户没有找到字符串，并终止替换（不会在下一行继续查找替换）。

11.5.7　区域性查找替换

在上一小节中，介绍了 Vim 编辑器中的查找替换功能。但所介绍的查找替换功能仅能替换光标所在行的第 1 个字符串。在本小节中，将介绍如何扩展查找替换的范围。

在第 5 章的 sed 命令中，介绍了一些指定区域的表示方法，例如 n,m 表示从第 *n* 行到第 *m* 行之间等。这些表示方法在 Vim 编辑器中也可以使用，下面将通过一些实例介绍如何在指定区域查找替换。

（1）通常替换命令只会替换在当前行内找到的第 1 个字符串，如果要替换当前行中找到的所有字符串（即全行替换），可以使用标记 g：

```
:s/the/The/g
```

（2）也可以指定一个要替换的范围。例如要替换第 1 行至第 10 行内的所有内容，使用如下命令：

```
:1,10s/the/The/g
```

（3）指定范围时，使用符号“%”表示所有行，用“$”表示文本的最后一行。例如要替换第 5 行至最后一行的所有字符串，使用如下命令：

```
:5,$s/the/The/g
```

（4）也可以使用“+”、“-”表示一个模糊的范围。例如要替换当前行以下至第 10 行和倒数第 10 行内的字符串，使用如下命令：

```
:+10,$-10s/the/The/g
```

（5）如果要替换所有字符串，使用如下命令：

```
:%s/the/The/g
```

在上面这个例子中，使用标记 g 的目的是替换每一行中的所有字符串。

上面简单列举了几个指定区域查找替换的例子，读者可以通过上面这几个例子，举一反三地进行练习。

11.5.8　谨慎的查找替换

如果当前文本中需要查找替换的字符串很多，并且字符串分布的区域未知，通常建议使用全文替换（即替换文件中的所有字符串）。但也有可能并非所有的字符串都要替换，这时就需要谨慎的查找替换。

执行谨慎的查找替换时，需要使用标记 c。例如：

```
:%s/the/The/gc
```

执行上面的命令后，Vim 将会高亮显示找到的字符串，并提示用户，如图 11.17 所示。

在图 11.17 的最下面，Vim 询问用户应该如何处理当前找到的字符串。此时可供选择的几个选项的含义如下。

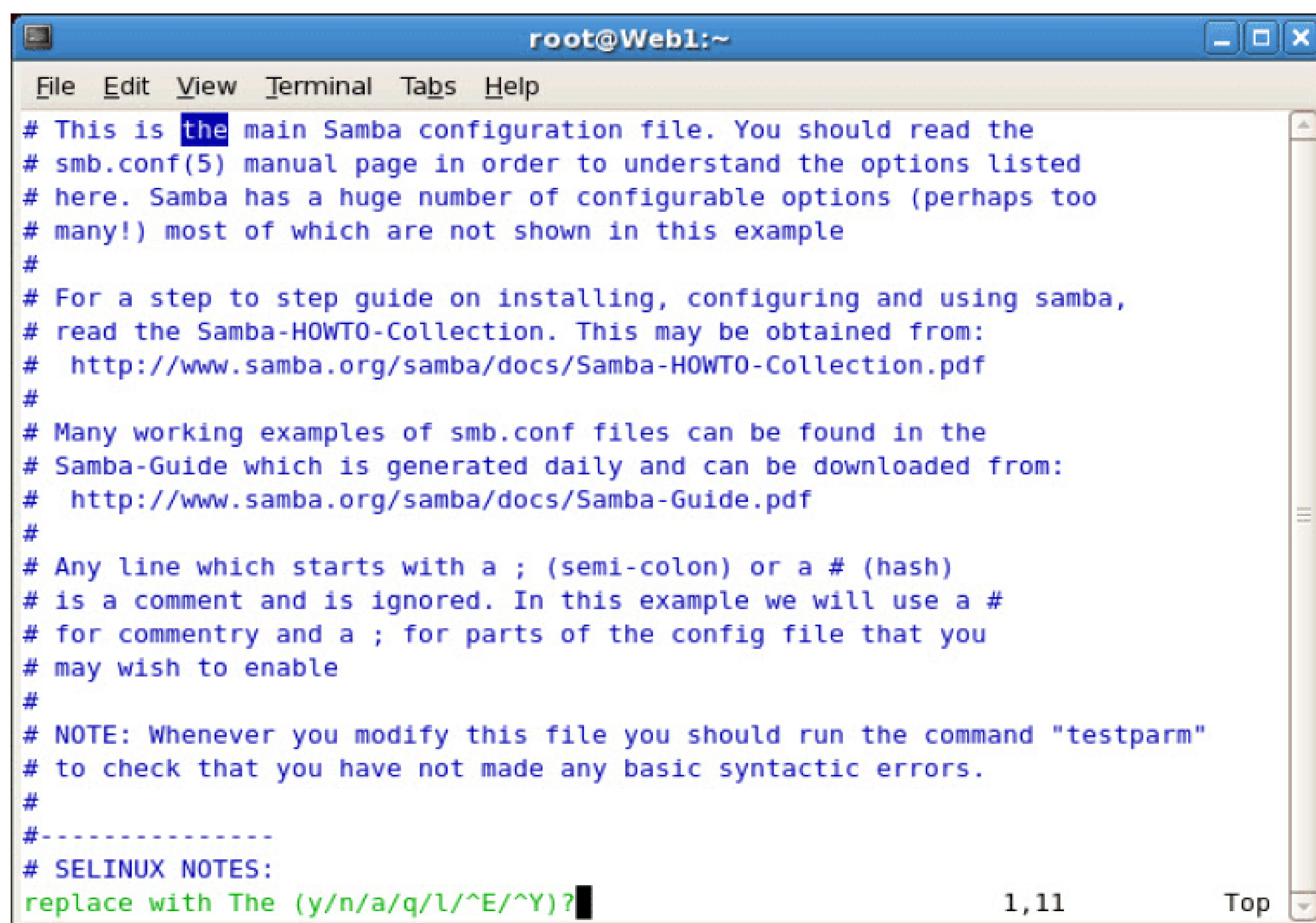


图 11.17　Vim 的提示信息

- ❑ y：直接输入 y 执行替换，继续查找字符串。
- ❑ n：直接输入 n 跳过当前找到的字符串，继续查找字符串。
- ❑ a：替换所有找到的字符串，并且不询问。
- ❑ q：不执行替换并退出替换模式。
- ❑ l：执行替换并退出替换模式。
- ❑ Ctrl+E：向上滚屏一行。
- ❑ Ctrl+Y：向下滚屏一行。

用户按需要选择快捷键进行操作即可。

11.6　Vim 编辑器中的窗口操作

有时可能更希望能在 Vim 编辑器中打开多个窗口同时编辑多个文本，特别是在使用一个大屏幕显示器时。本节将简单介绍如何在 Vim 编辑器中开启多个窗口，并在不同的窗口中编辑不同的文本。

11.6.1　分割窗口

在 Vim 编辑器中，打开窗口是以分割当前窗口的形式完成的。按功能可以将分割窗口

分为：分割当前窗口、垂直分割、分割窗口后打开新文件等，本小节将简单介绍分割窗口的方法。

1．分割当前窗口

先使用 Vim 编辑器打开一个文本文件，然后使用“:split”分割当前窗口。分割后的 Vim 编辑器如图 11.18 所示。

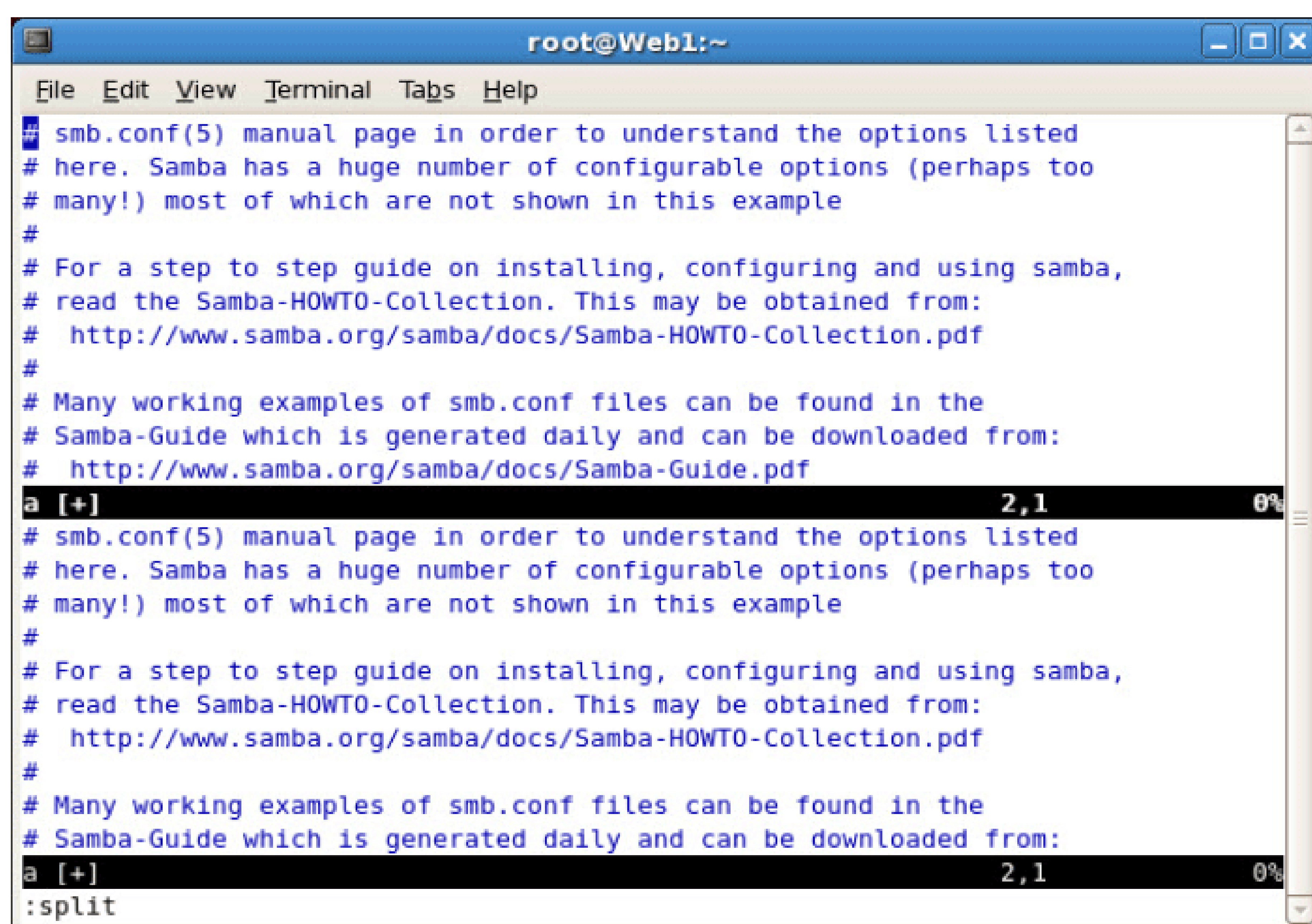


图 11.18　分割后的 Vim 编辑器

从图 11.18 中可以看出，使用分割命令之后，Vim 编辑器被分割为上下两个窗口，并且之前打开的文本文件会同时在两个窗口中显示。将 Vim 编辑器分割为两个窗口后，Vim 会将光标放在第 1 个窗口中。

2．垂直分割

如果觉得将窗口分为上下两个窗口不方便，还可以使用垂直分割。垂直分割的命令是“:vsplit”，效果如图 11.19 所示。

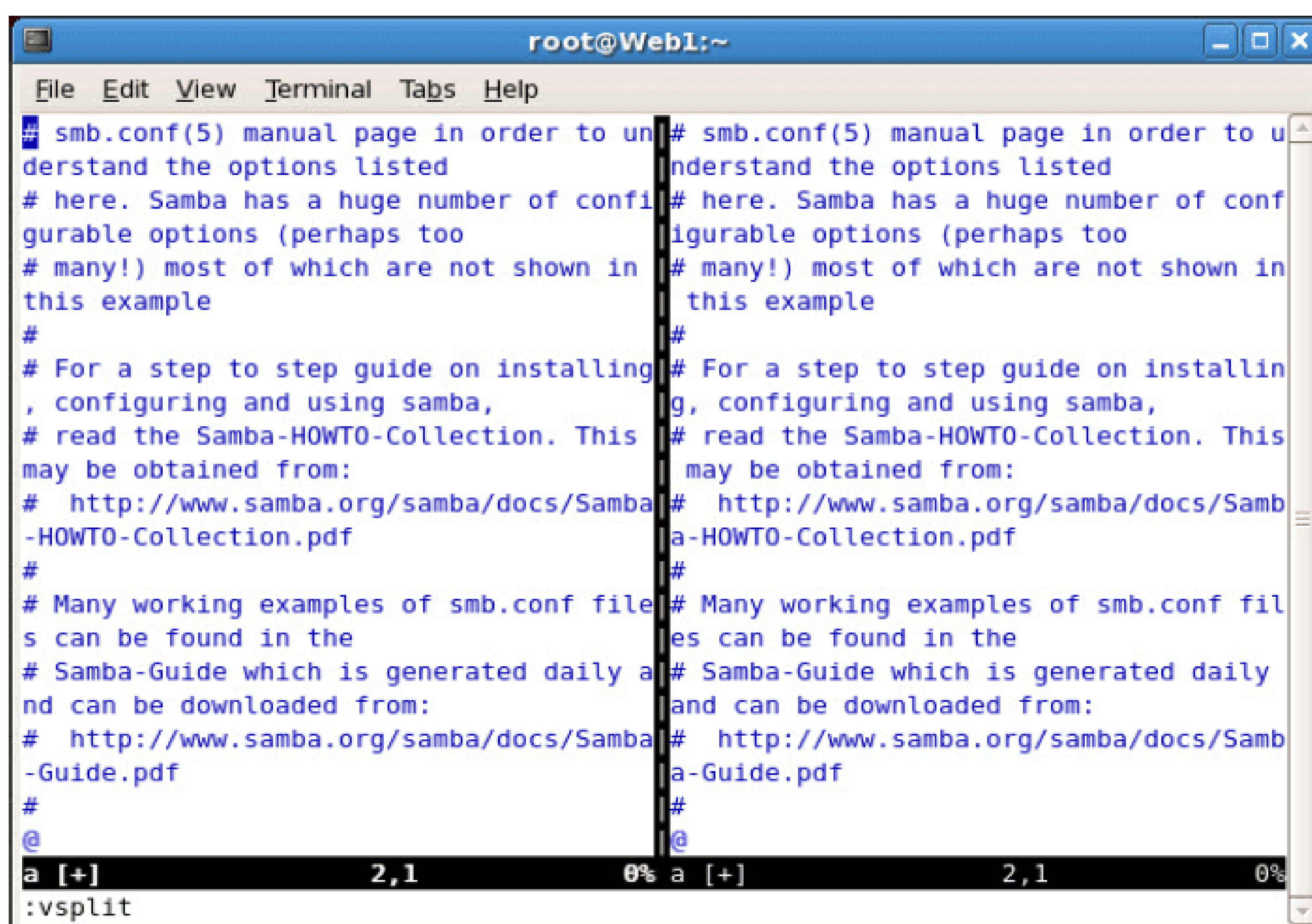


图 11.19　垂直分割示意图

从图 11.19 中可以看出，使用了“:vsplit”命令之后，Vim 编辑器被分割为左右两个窗口，并将光标放在左边的窗口中。

3．分割新窗口并打开空白文本

有时可能需要在 Vim 编辑器中打开新窗口并打开空白文本，这时可以使用命令“:new”。分割新窗口并打开空白文本的效果如图 11.20 所示。

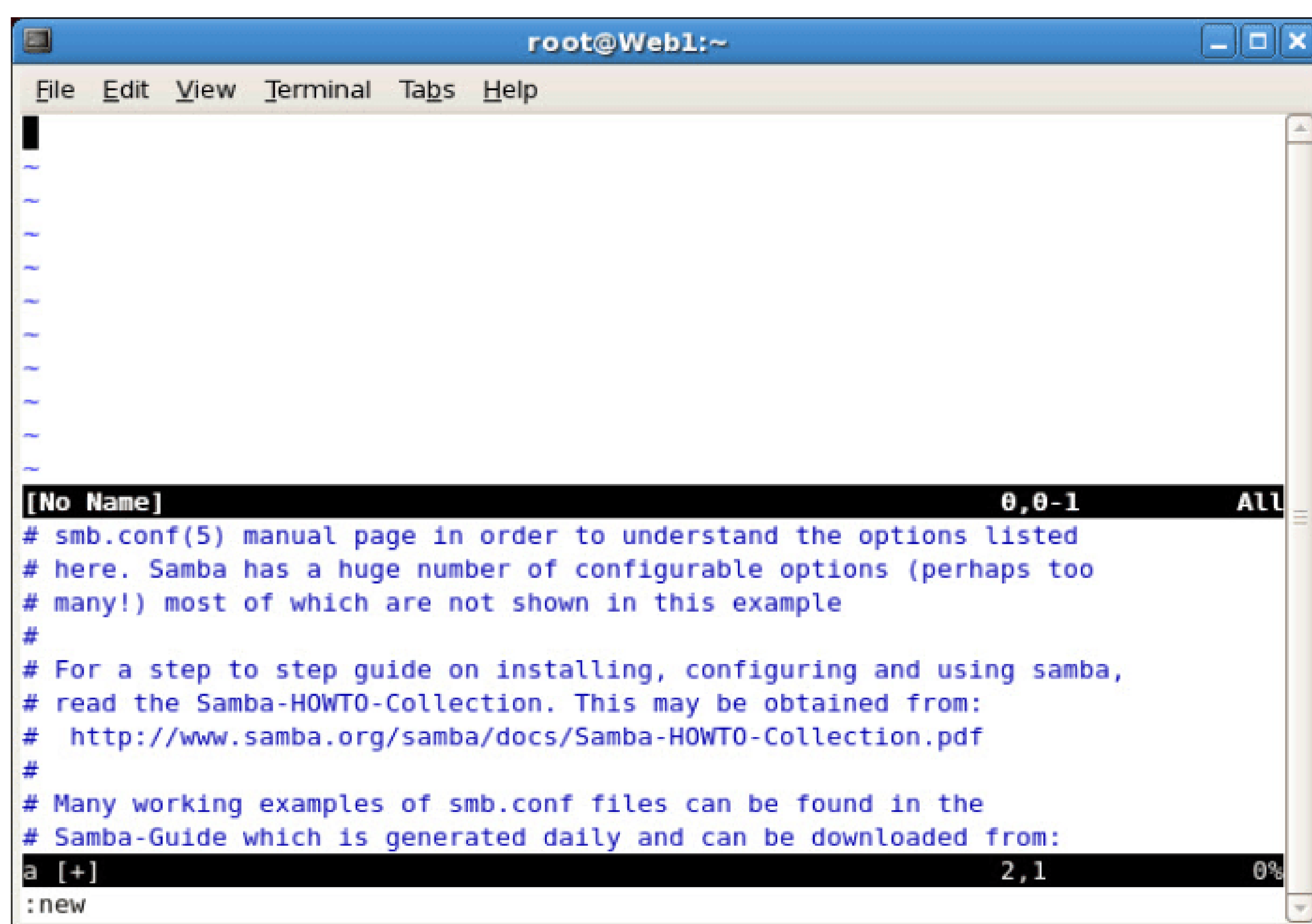


图 11.20 分割新窗口并打开空白文本后的 Vim 编辑器

从图 11.20 中可以看出，Vim 编辑器打开的空白文本没有命名（状态栏中标识为“[No Name]”），并且新窗口位于编辑器窗口的上部。

4．分割窗口并打开新文件

如果需要分割窗口并在新窗口中打开新文件，可以在命令“:new”后面加上要打开的文件名。例如使用命令“:new /etc/ssh/sshd_config”进行分割之后，Vim 编辑器的界面如图 11.21 所示。

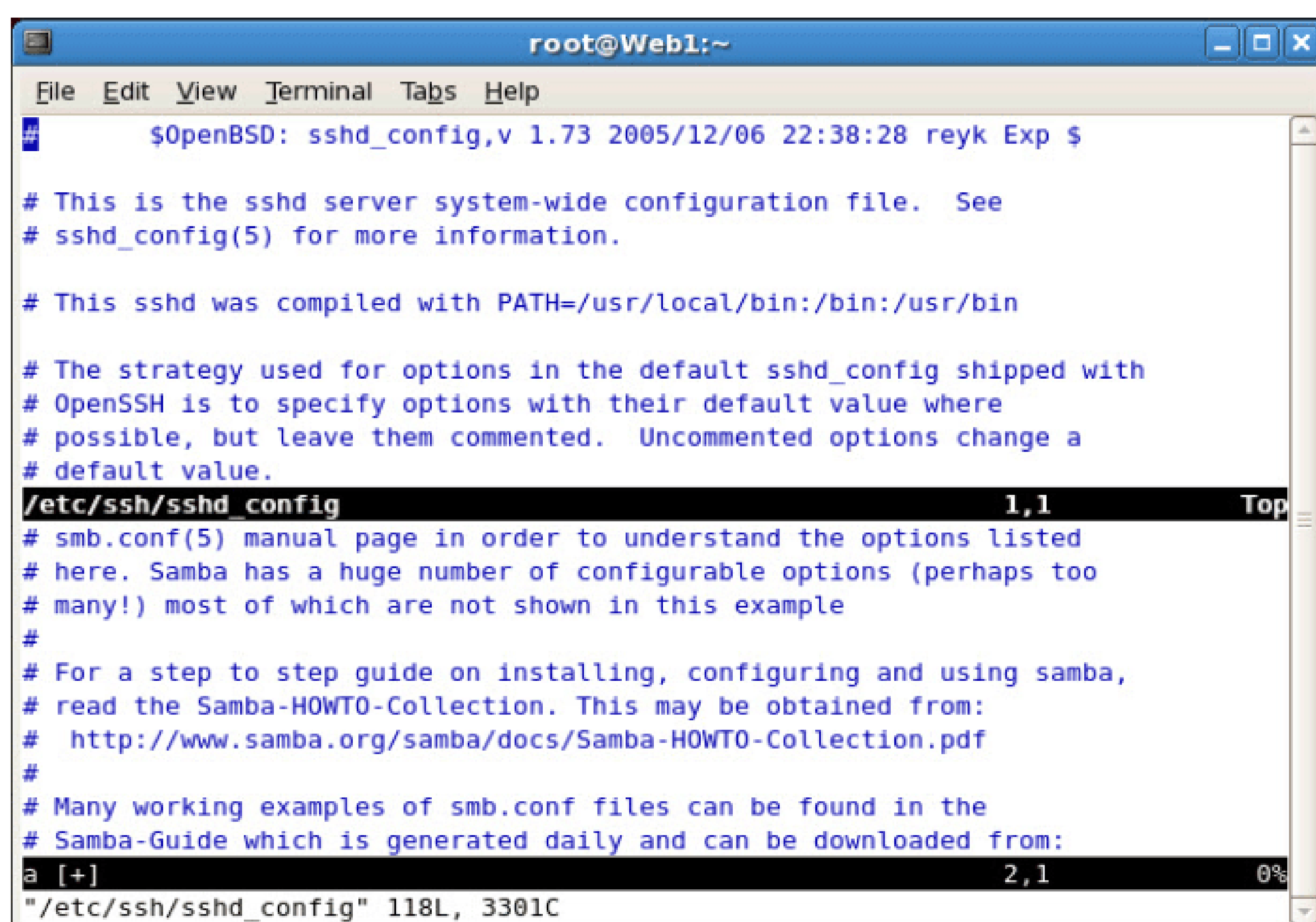


图 11.21 分割新窗口并打开文件

在图 11.21 中可以看出，新分割的窗口位于编辑器的上部，并且 Vim 编辑器会将光标移动到新窗口中。

使用本小节中介绍的分割命令时需要注意，除了命令“:new”外，其他命令也可以接受文件参数。

技巧：如果需要在已经分割的窗口中继续分割，可以在相应的窗口中继续使用分割命令。

5．打开文件时分割窗口

如果要在打开文件时一并打开多个窗口，在不同的窗口中打开不同的文件，可以在启动 Vim 时使用选项 o。例如执行命令 vim –o /etc/samba/smb.conf /etc/ssh/ssh_config /etc/ssh/ssh_config，Vim 启动后会在 3 个窗口打开不同的文件，如图 11.22 所示。

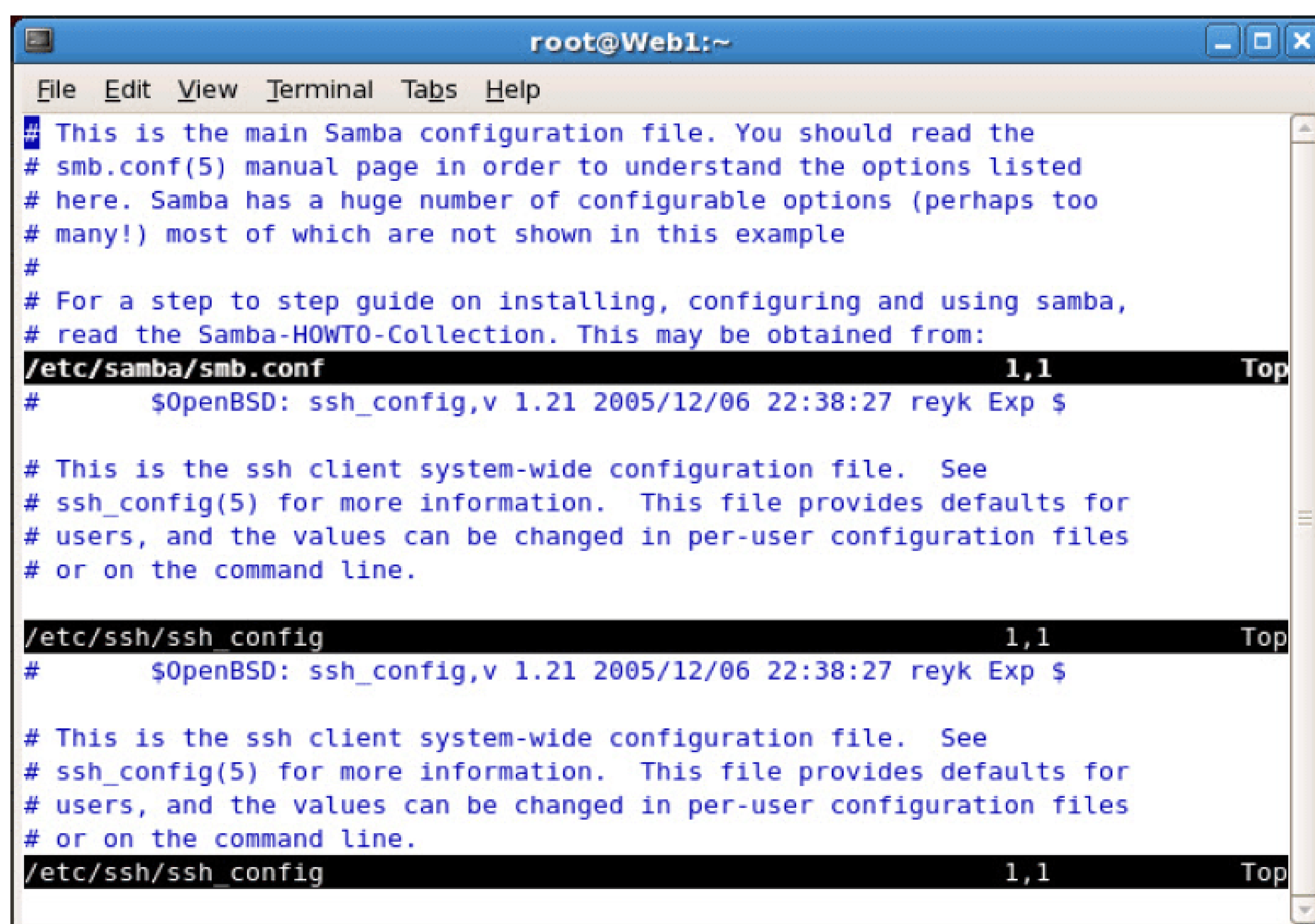


图 11.22

11.6.2　关闭窗口

将 Vim 编辑分割为多个窗口之后，可能需要关闭已经开启的窗口。本节将简单介绍如何关闭 Vim 编辑器的窗口。

（1）如果要关闭光标所在窗口（如果光标不在需要关闭的窗口中，可以使用快捷键 Ctrl+W），可以在当前窗口的命令模式中使用命令“:close”。

（2）如果需要关闭除当前窗口之外的其他窗口，可以在当前窗口的命令模式中使用命令“only”。

11.6.3　控制窗口大小

如果在 Vim 编辑器中打开了好几个窗口，但不使用所有窗口，将不使用的窗口缩小是一个好办法，本小节将简单介绍如何控制窗口的大小。

（1）要控制分割的窗口大小，可以在分割窗口时加上窗口大小参数。例如分割一个只有 6 行的新窗口：“:6split”、“:6vsplit”、“:6new”……

（2）如果要将当前光标所在窗口放大，可以使用快捷键 Ctrl+W++（即 Ctrl+W+Shift+=）。

（3）如果要缩小窗口，可以使用快捷键 Ctrl+W+-。

无论使用什么方法控制窗口大小，放大、缩小窗口时，都只能将窗口放大或缩小到窗口无法再放大或缩小时为止。

11.6.4　窗口中的操作

将 Vim 编辑器分割为多个窗口之后，难免需要在窗口中进行操作。大多数情况下每个窗口的操作都与单窗口的操作相同，本小节将简单介绍多窗口中的不同操作。

1．切换窗口

如果使用了多次分割命令，将 Vim 编辑器分割为多个窗口，使用 Ctrl+W 命令切换窗口可能就力不从心了。这时可以尝试使用如下快捷键。

- ❑ Ctrl+W k：将光标移动到上面的窗口中。该组合键的使用方法是：先按下 Ctrl 键和 w 键，然后松开 Ctrl 和 w 键，最后再按下 k 键。
- ❑ Ctrl+W j：将光标移动到下面的窗口中。
- ❑ Ctrl+W h：将光标移动到左边的窗口中。
- ❑ Ctrl+W l：将光标移动到右边的窗口。
- ❑ Ctrl+W t：移动光标至顶部的窗口中。
- ❑ Ctrl+W b：移动光标至底部的窗口中。

读者可能会觉得这几个快捷键有些眼熟，不错，k、j、h、l 正是命令模式中移动光标的快捷键。

2．从众多窗口中退出

在打开的窗口中一一使用退出命令，可能会比较繁琐。这时可以尝试使用以下命令退出。

- ❑ qall：使用此命令后，Vim 编辑器会尝试关闭所有窗口并退出。如果有的窗口中的修改还没有保存，窗口将不会关闭，对于这类窗口，用户可以单独保存或退出。此命令可速记为“quit all”。
- ❑ wall：保存所有修改过的窗口，可速记为“write all”。
- ❑ wqall：保存并退出所有窗口，可速记为“write quit all”。
- ❑ qall!：不保存未修改的内容，强制关闭所有窗口，并退出 Vim。如果还有未保存的文件，使用此命令要格外小心。可速记为“quit all!”。

由于每个窗口中都可能编辑不同的文件，因此从众多窗口中退出时，一定要格外谨慎，以免丢失数据。

11.6.5　移动窗口

有时可能对 Vim 编辑器打开的多个窗口的位置并不满意，这时可以对这些窗口的位置

进行调整。本小节将简单介绍如何移动 Vim 编辑器中的窗口。

在 Vim 编辑器中，移动窗口可以使用以下几个快捷键。

- Ctr+W K：将当前光标所在窗口向上移动。
- Ctr+W J：将当前光标所在窗口向下移动。
- Ctr+W H：向左移动窗口。
- Ctr+W L：向右移动窗口。

使用上面这几个快捷键时需要注意，K、J、H、L 键都是大写。此处不再赘述这几个快捷键的用法，读者可以打开多个窗口自行练习。

11.7　Vim 编辑器的高级技巧

在本章的前面几节中，介绍了 Vim 编辑器的基本界面、使用方法和技巧。本节将在前几节的基础上介绍 Vim 编辑器的其他技巧。

11.7.1　复制和粘贴

在图形界面环境中的文本编辑器中，往往提供了 Ctrl+C 和 Ctrl+V 快捷键执行复制和粘贴操作。Vim 的命令模式也提供了几个复制、粘贴操作的快捷键。

- yy：复制光标所在的行至缓冲区内。
- *n*yy：复制 *n* 行至缓冲区内。
- y^：复制当前光标所在位置到行首的内容至缓冲区内。
- y$：复制当前光标所在位置至行尾的内容至缓冲区内。

使用了以上快捷键执行复制操作之后，只需要将光标移动到需要粘贴的位置，按下 p 键即可粘贴复制的文本。

11.7.2　剪切和粘贴

在 Vim 编辑器中没有单独的剪切、粘贴快捷键，但可以通过移动文本的方式实现剪切、粘贴功能。

（1）移动文本时，首先需要使用 dd 或 d 快捷键将需要移动的文本删除，Vim 会在删除之后将删除的文本放入缓冲区中。

（2）执行删除之后，将光标移动到需要粘贴的位置，然后使用 p 键粘贴文本。

（3）执行 p 键后，Vim 编辑器会将缓冲区中的内容粘贴到当前光标之后（如果使用 dd 快捷键删除，则粘贴至下一行）。

技巧：粘贴文本除了使用 p 键之外，还可以使用 P 键（大写字母），其功能是将缓冲区中的文本粘贴到当前光标之前。

11.7.3　编辑多个文件

虽然 Vim 编辑器可以在多个窗口中打开多个不同的文件，但打开的文件窗口都比较

小，可能会影响编辑的效率。幸运的是 Vim 编辑器也提供了同时编辑多个文件的功能。本小节将简单介绍如何编辑多个文件。

（1）例如要编辑文件 a1、a2 和 a3：

```
#使用 vim 命令打开多个文件
$ vim a1 a2 a3
```

虽然可以打开多个文件，但同一时间用户只能编辑一个文件。上面这个示例命令打开多个文件后，Vim 默认将文件 a1 放置在前台供用户编辑。

（2）如果 Vim 打开了多个文件，可以使用以下命令查看打开的文件列表：

```
:args
```

执行了以上命令后，编辑器将在最后一行显示当前打开的文件列表，并将处于前台的文件名放在大括号“[]”内，如图 11.23 所示。

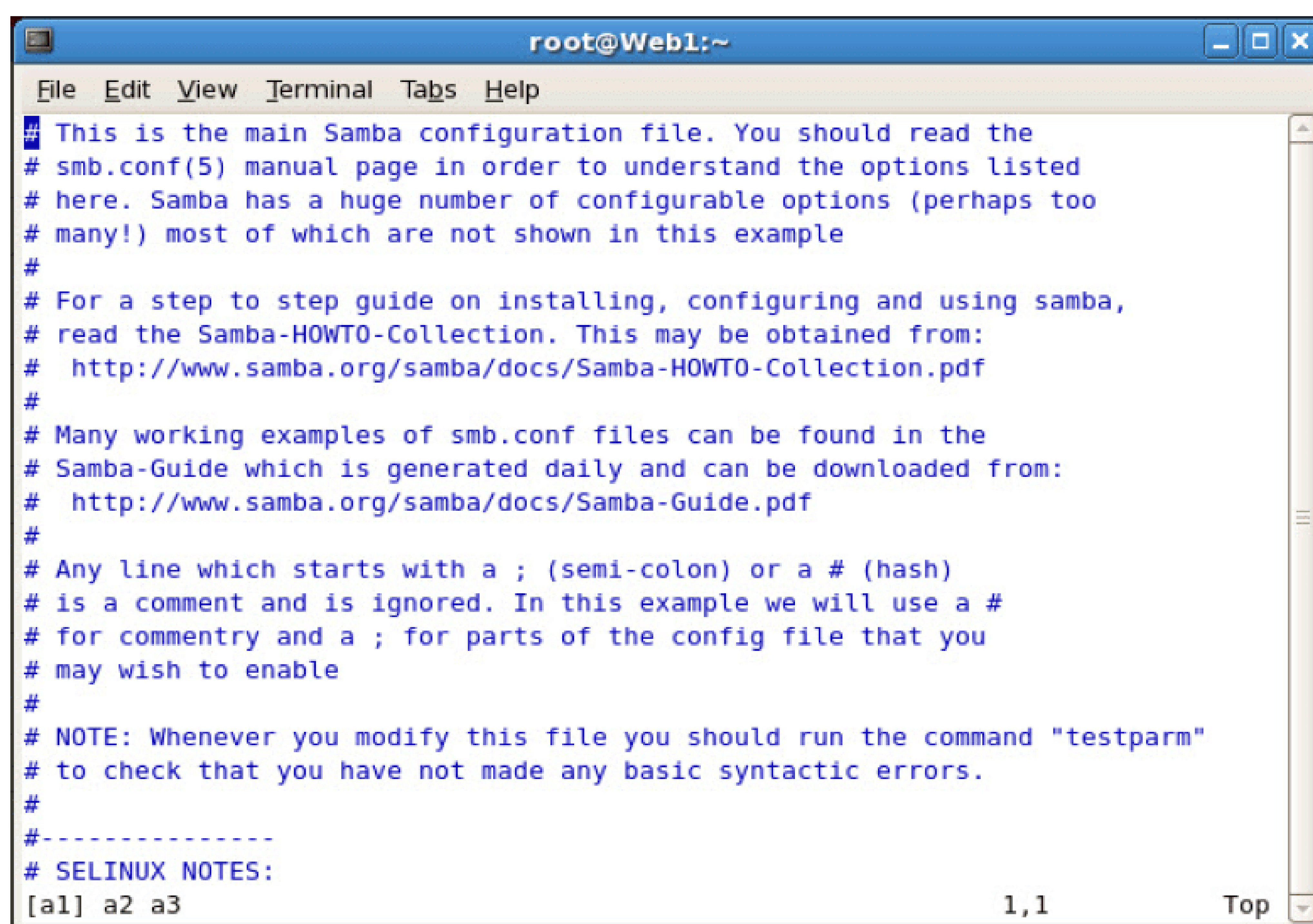


图 11.23　Vim 的多文件编辑

在图 11.23 中，底部的状态栏显示当前正在编辑 3 个文件，文件名为：a1、a2、a3。“[a1]”表示当前编辑器中正在编辑的是文件 a1。

（3）如果要在打开的多个文件之间切换，首先应该确保当前处于前台的文件已经保存，然后可以使用以下几个命令和快捷键切换前台文件。

- next：将下一个文件放置到前台。
- prev：将上一个文件放置到前台。
- next!：不保存当前文件，将下一个文件放置到前台。
- prev!：不保存当前文件，将上一个文件放置到前台。
- first：将第 1 个文件放置到前台。
- last：将最后一个文件放置到前台。
- first!：不保存当前文件，将第 1 个文件放置到前台。
- last!：不保存当前文件，将最后一个文件放置到前台。
- Ctrl+6：在最近放置到前台的两个文件之间切换。

如果正在编辑多个文件，使用退出编辑器命令 q，编辑器将会关闭所有打开的文件并退出，而不是退出单个打开的文件。

11.7.4　Visual 模式

在对文本执行编辑时可能会觉得不方便，例如要删除多行时，需要计算具体有多少行，然后再使用 dd 快捷键，否则就只能使用多次 dd 快捷键；要删除多个字符时，只能不停地按下删除快捷键 d。遇到这种情况时，可以考虑使用 Vim 编辑器的 Visual 模式。本小节将简单介绍如何使用 Visual 模式。

1．使用Visual模式自由选择文本

使用 Visual 模式自由选择字符时，需要在命令模式中将光标移动到选择字符的起始位置，然后按下 v 键进入 Visual 模式。在 Visual 模式中，可以使用移动光标的快捷键选择文本，如图 11.24 所示。

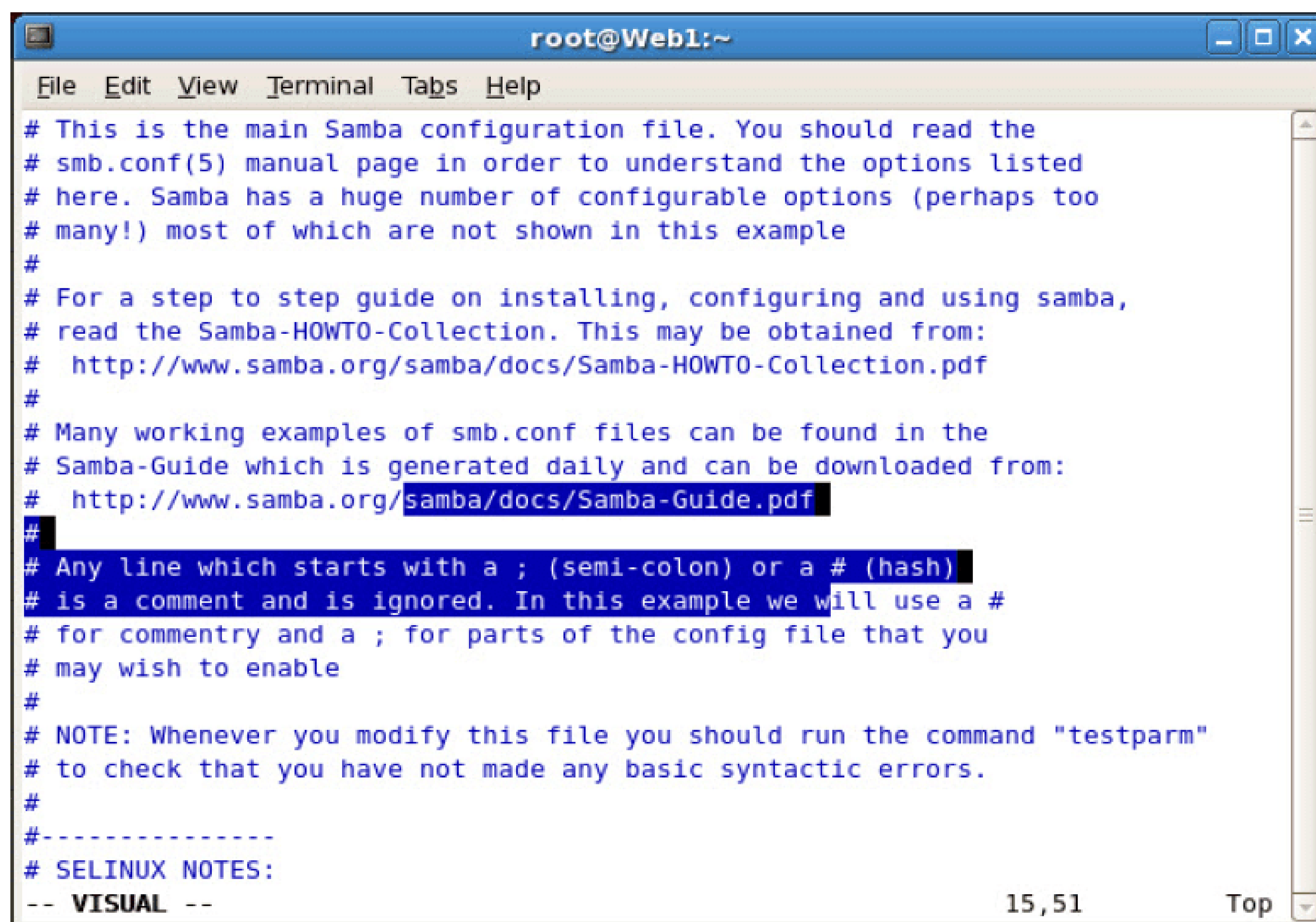


图 11.24　在 Visual 模式中自由选择文本

在图 11.24 中，Vim 在状态栏中使用“-- VISUAL --”提醒用户当前处于 Visual 模式，并将选中的文本以蓝底白字的形式显示出来。在 Visual 模式中选中需要的文件后，就可以使用复制、删除等快捷键了。执行完快捷键后，Vim 将会自动退出 Visual 模式，并返回命令模式。

技巧：如果需要从 Visual 模式中退出，可以按下 Esc 键，该操作不会引起任何文件内容上的变化。

2．使用Visual模式选择多行

在 Visual 模式中选择多行时，需要先在命令模式中将光标移动到起始位置，然后按下 V 键进入 Visual 模式。在 Visual 模式中，可以使用上、下方向键或 j、k 键选择文本行，如

图 11.25 所示。

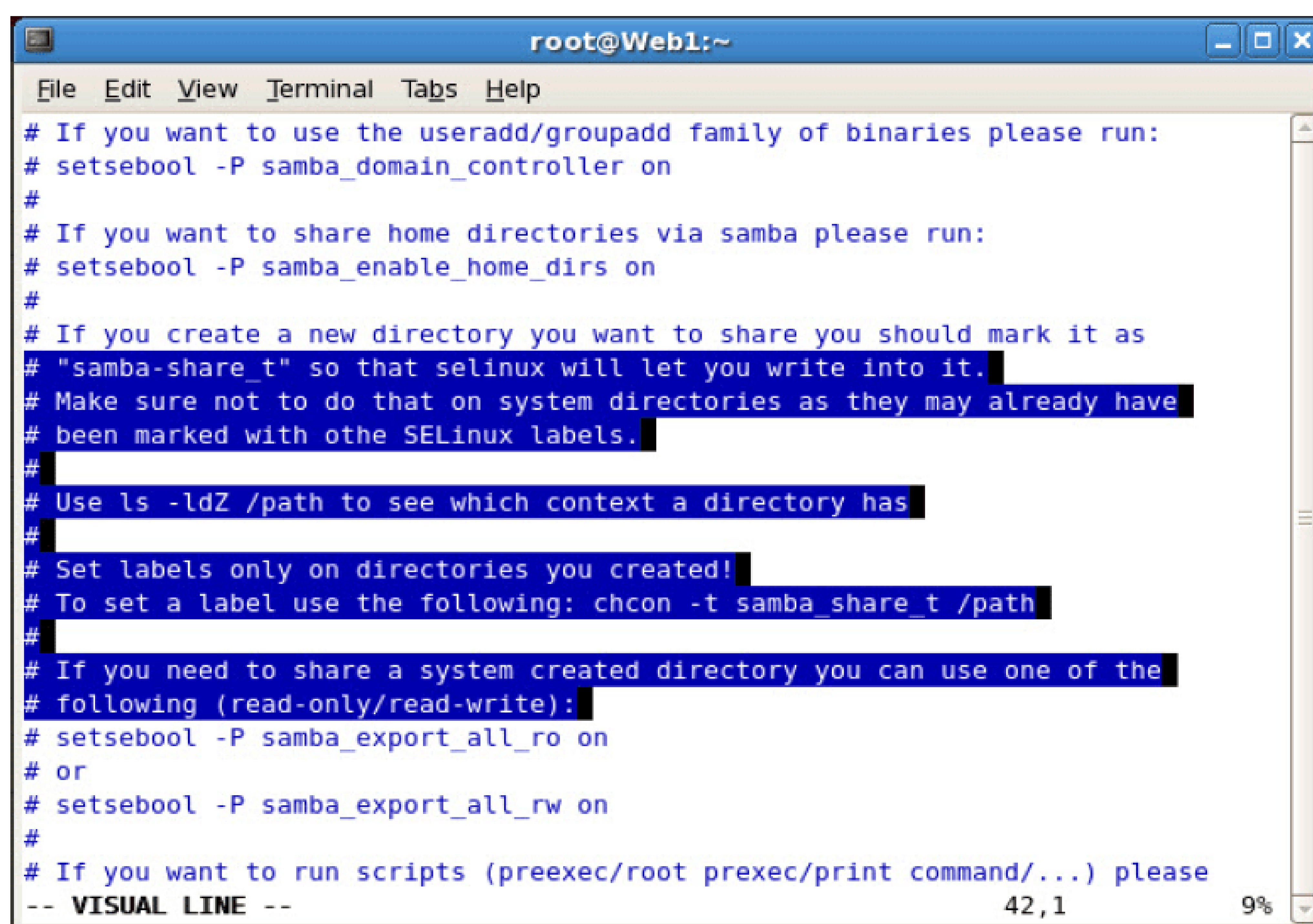


图 11.25　在 Visual 模式中选择多行

在图 11.25 中，Vim 在状态栏中使用“-- VISUAL LINE --”提示用户当前处于 Visual 行模式，并以蓝底的形式显示用户选中的文本。

与前面的自由选择一样，选择完文本后，就可以执行复制、删除等快捷键了，Vim 将在执行完快捷键后自动返回命令模式。

3. 使用Visual模式选择矩形区域

在 Visual 模式中，还可以选择一个矩形区域，这主要用于选择格式化文本。例如一个销售报表、学生名单等。选择矩形区域时，先在命令模式中将光标移动到矩形区域的左上角的顶点，然后按下快捷键 Ctrl+V 进入 Visual 模式。在 Visual 模式中，使用方向键移动光标即可选择一个矩形区域，如图 11.26 所示。

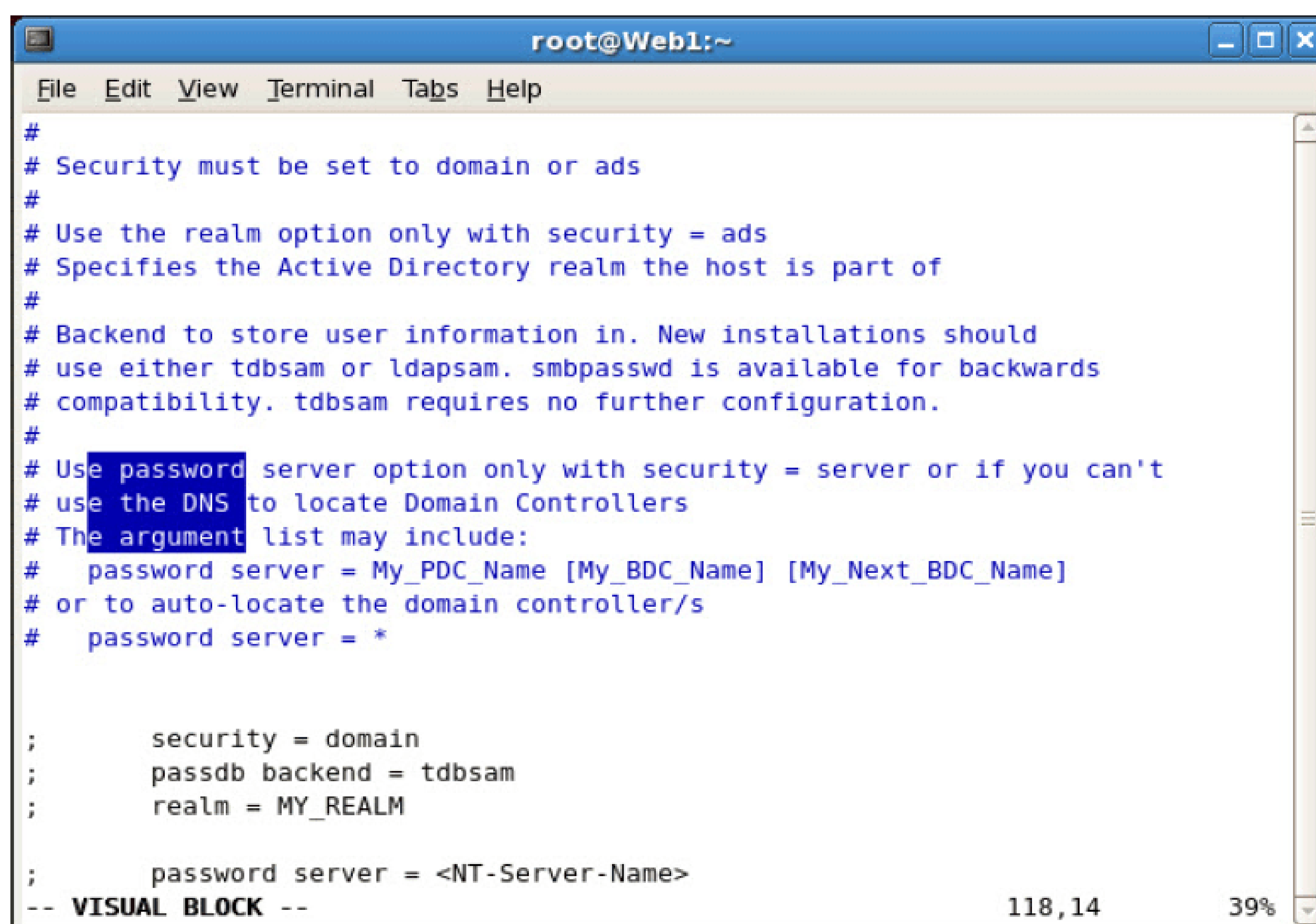


图 11.26　在 Visual 模式中选择矩形区域

在图 11.26 中，Vim 编辑器在状态栏中用“--VISUAL BLOCK --”提示用户当前正在选择一个矩形区域。与之前的自由选择和多行选择相同，选择并执行快捷键后，Vim 编辑器会自动退出 Visual 模式返回命令模式。

技巧： 在 Visual 模式中选择文本时，如果要修改文本的起始位置，可以使用快捷键 o（注意区别命令模式中的快捷键 o）切换至起始位置重新选择。

11.7.5　在 Vim 编辑器中执行 Shell 命令

有时需要在 Vim 编辑器中执行 Shell 命令，例如需要验证一个 Shell 命令是否正确，以便写入脚本中；需要在文件中引用某个 Shell 命令的输入等。本小节将简单介绍如何在 Vim 编辑器中执行 Shell 命令。

（1）在 Vim 编辑器中执行单独的 Shell 命令时，需要在命令之前加上“:!”。例如在 Vim 命令模式中执行 Shell 命令“:!ls -l”，Vim 编辑器将会显示命令的输出，如图 11.27 所示。

```
root@Web1:~
File  Edit  View  Terminal  Tabs  Help
[root@Web1 ~]# vim a

total 2199776
-rw-r--r-- 1 root root       9733 May 31 04:55 a
-rw-r--r-- 1 root root       9733 Sep 29 16:50 a1
-rw-r--r-- 1 root root         17 Sep  7 05:17 a2
-rw-r--r-- 1 root root         18 Sep  7 09:18 a3
-rw------- 1 root root       1275 May  6 01:45 anaconda-ks.cfg
drwxr-xr-x 3 root root       4096 Aug 11 19:35 apm_install
-rw-r--r-- 1 root root  208803840 Jul 31 00:36 apm_install.tar
drwxr-xr-x 2 root root       4096 Sep  3 14:53 Desktop
-rw-r--r-- 1 root root         20 Aug 27 14:55 example
-rwxr--r-- 1 root root         84 Aug 30 23:17 Hello.csh
drwxr-xr-x 9 root root       4096 Sep  1 05:46 initrd
-rw-r--r-- 1 root root      29989 May  6 01:45 install.log
-rw-r--r-- 1 root root       3543 May  6 01:41 install.log.syslog
-rw-r----- 1 root root 2040659968 Aug 29 16:54 lenovo_B460.iso
-rw-r--r-- 1 root root          4 Sep  4 03:27 master_status.log
-rw------- 1 root root      88723 Sep  2 14:58 mbox
-rw-r--r-- 1 root root     603609 Nov 22  2010 nano-2.2.6-1.i386.rpm
drwxr-xr-x 2 root root       4096 Sep  7 04:33 test
-rw-r--r-- 1 root root       9733 May 31 04:56 test1

Press ENTER or type command to continue
```

图 11.27　在 Vim 编辑器中执行 Shell 命令

从图 11.27 中可以看出，Vim 将会显示命令的输出。如果用户需要返回编辑文件，可以按下 Enter 键。

（2）有时可能需要在编辑器编辑的文件中引用 Shell 命令的输出，这时可以配合命令“:r”。例如读取 ls-l 命令的输出到当前编辑的文件中，可以使用“:r !ls -l”。这时 Vim 编辑器会读取命令的输出，并将其插入到当前编辑的文件中。

11.8　定制 Vim 编辑器及灾难恢复

使用 Vim 编辑器编辑文本时，Vim 编辑器会使用默认值启动。然而这并不能适合所有人的需要，可能有些人需要对 Vim 编辑器进行定制，使其一启动就能够使用自己需要的设

置，这是一个非常好的办法。本节将介绍定制 Vim 编辑器及灾难恢复。

11.8.1　定制文件 vimrc

有些用户可能希望 Vim 编辑器一启用就能展示自己经常使用的功能，例如显示行号、搜索结果高亮显示等功能，这时就需要定制 Vim 编辑器。本小节将简单介绍如何定制 Vim 编辑器。

【Vim 编辑器的定制文件】

定制 Vim 编辑器主要是通过修改定制文件的方法来实现的。Vim 编辑器使用的定制文件名称为.vimrc，但该文件在不同的系统中位置和名称可能会不同。下面是 Vim 在不同系统中保存的定制文件的位置及名称。

- UNIX：文件位置及名称为$HOME/.vimrc。
- OS/2：文件位置为$HOME 或$VIM，文件名称通常为.vimrc 或_vimrc。
- MS-DOS 和 Windows：文件位置及名称为$HOME/_vimrc 或$VIM/_vimrc。

在 Linux 系统中，定制文件的路径及名称通常为$HOME/.vimrc。如果无法找到定制文件，可以使用以下命令查看帮助：

```
:help vimrc
```

11.8.2　定制 Vim 编辑器

定制文件的内容主要是常用的命令，包括搜索结果显示高亮、增量查找、显示行号、设置快捷键及语法检查等。在定制文件中使用命令时，不需要在命令前使用“:”，直接输入要使用的命令即可。

下面是一个简单的定制文件的示例：

```
#使用 cat 命令查看定制文件的内容
#如果用户的家目录中没有此文件，可以自己新建一个
# cat ~/.vimrc
"    File: ~/vimc
"    Author: Wg

" Display line number
set number

" Highlight search thing and increase
set hlsearch
set incsearch=100
```

在 Vim 编辑器的定制文件中，使用双引号“"”开头的行表示一个注释行，Vim 编辑器在读取定制文件时会自动忽略这些注释行。

除此之外，在定制文件中还可以使用函数、流程控制等类似于程序语言的语句，关于这些命令及用法，感兴趣的读者可以参考相关文档，此处不再赘述。

注意：除了使用定制文件对 Vim 编辑器进行定制以外，还可以使用 plugin 扩展对 Vim 编辑器进行定制，感兴趣的读者可以阅读相关文档。

11.8.3　灾难恢复

当用户正在编辑文件时，一些意外的情况可能会导致编辑的文本丢失，例如主机断电、编辑器崩溃等。此时可以使用 Vim 编辑器保存的交换文件恢复一些数据。但恢复数据的前提是硬盘还可以读写。如果硬盘被损坏，那么可能只有到专业数据恢复实验室处理了。

使用 Vim 编辑文本文件时，会自动建立一个交换文件，名称为.filename.swp。这是一个隐藏文件，其中的 filename 为编辑的文件名称。每过一段时间，Vim 会自动将更改的内容保存到交换文件中。Vim 编辑器正常退出时，会删除这个交换文件，如果是非正常退出，这个文件就留在了文件系统上，此时可以利用交换文件进行灾难恢复。

（1）大多数时候进行灾难恢复非常简单，只需要配合使用选项 r 即可。例如要恢复的文件名为 a：

```
#使用选项 r 恢复文件 a 的内容
# vim -r a
```

执行上述命令之后，建议先将恢复后的文件重命名，对文件内容进行比较和确认之后，再进行保存。

（2）在恢复文件时，也可以指定交换文件，例如要使用名为.a.swp 的交换文件进行恢复，可以使用如下命令：

```
#指定用于恢复的交换文件名称
# vim -r .a.swp
```

（3）如果要恢复的文件还没有命名，可使用如下命令：

```
#如果文件还未命名时，可以使用两个引号表示文件名为空
# vim -r ""
```

（4）有时使用 Vim 编辑器打开一个文件会出现如图 11.28 所示界面。

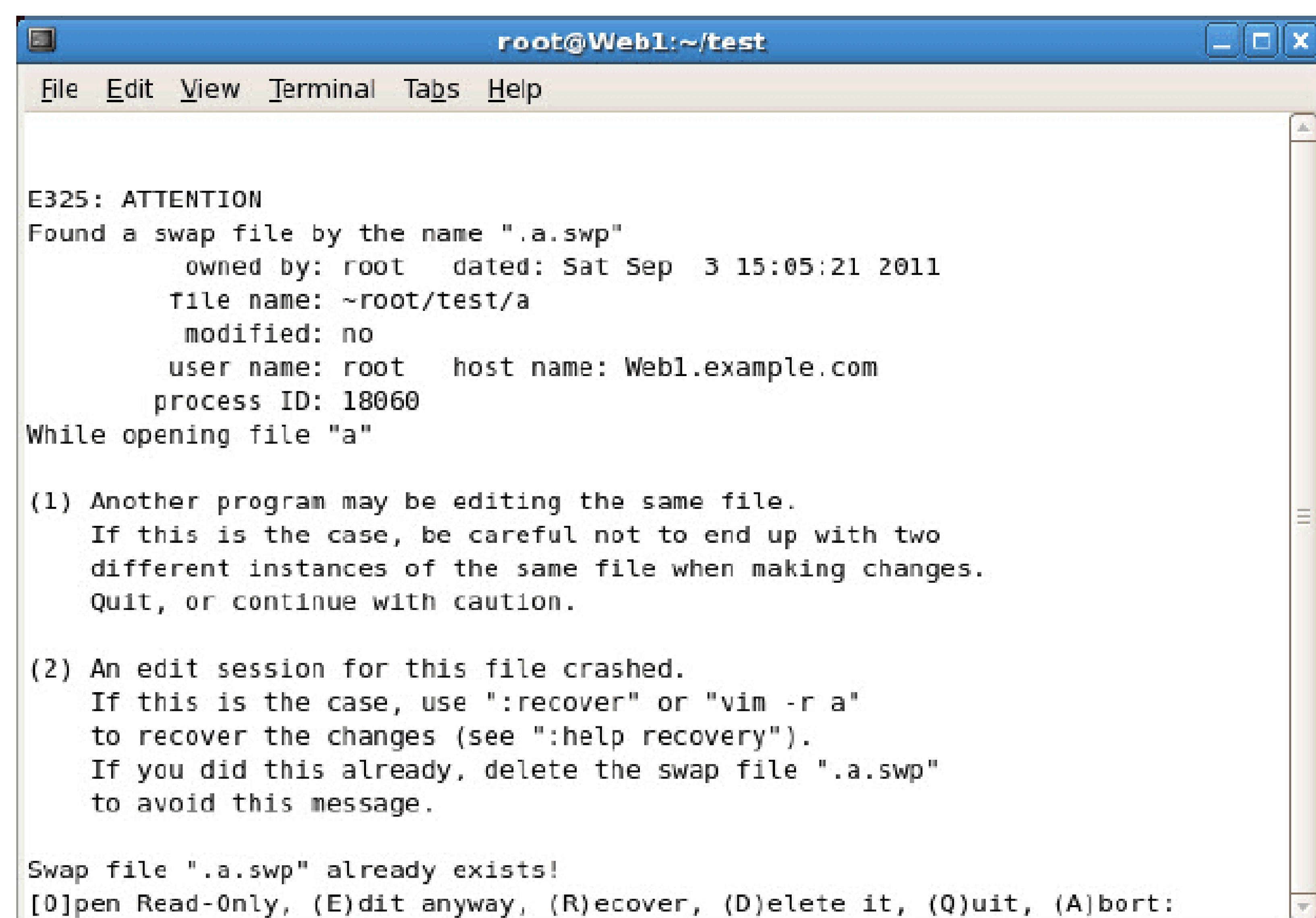


图 11.28　Vim 编辑器找到交换文件

图 11.28 表示 Vim 编辑器找到了当前编辑文件的交换文件，可能是由于上次非正常退出造成的，也可能是由于该文件正在被另一个用户或另一个会话编辑（参见 process ID: 18060 的提示）。

出现以上情况时编辑器会询问用户应该如何进行下一步操作，并给出了一些建议。此时常用的操作如下。

- ❑ 按 O 键以只读方式打开。如果另一个会话正在编辑该文件，而当前只是想查看该文件的内容，使用该操作。
- ❑ 按 E 键不恢复，继续编辑文件。如果另一个用户正在编辑该文本，应该格外小心，这个选项可能会造成该文本拥有两个不同的版本，如非必要，建议以只读方式打开，另行存档后再进行编辑。
- ❑ 按 R 键恢复上次没有保存的内容。使用此选项的前提是非常确定要恢复的内容就是需要找的内容。
- ❑ 按 Q 键退出。如果该文件正在被另一个用户或会话编辑，此选项将是最佳的选择。
- ❑ 按 A 键丢弃。如果此时正在使用 Vim 打开多个不同的文件，丢弃是最好的选择。
- ❑ 按 D 键删除交换文件。如果已经确认这些交换文件没有任何价值和意义，可以使用此项删除已存在的交换文件。

虽然 Vim 编辑器拥有灾难恢复功能，但并不是用户的每一次更改 Vim 都会保存到交换文件中，因此并不是所有编辑过的内容都能恢复。

本书仅简单介绍 Vim 灾难恢复的机制和简单应用。关于交换文件和灾难恢复的更多帮助，读者可以参考帮助主题 swap-file、swapname、:preserve 等内容，也可以参考互联网中的相关文档。

注意： Vim 编辑器的使用技巧非常多，本书中仅简单介绍管理员需要掌握的常见功能，感兴趣的读者可以参考相关文档了解更多知识。

11.9 小　　结

- ❑ 11.1 节主要介绍了文本编辑器的发展及其分类。
- ❑ 11.2 节简单讲解了许多发行版都自带的 Vi 编辑器的使用方法。
- ❑ 11.3 节简单介绍了 Vim 编辑器中的简单操作，包括如何读取、保存文件，编辑文件等内容。
- ❑ 11.4 节讲解了如何在 Vim 编辑器中快速移动光标。
- ❑ 11.5 节介绍了 Vim 编辑器中的查找功能、查找替换功能。
- ❑ 11.6 节简单介绍了 Vim 编辑器中窗口的使用、窗口的操作等。
- ❑ 11.7 节讲解了 Vim 编辑器中的高级技巧，包括复制、剪切、粘贴、同时编辑多个文件等内容。
- ❑ 11.8 节主要介绍了 Vim 编辑器的定制文件 vimrc 和灾难恢复功能。

由于目前大多数管理员都是通过远程终端方式使用 Linux 系统，因此学会并熟练地使用 Linux 系统中的字符界面文本编辑器十分重要。建议所有初学者都掌握 Vim 的基本用法，以便需要时使用（例如在救援模式中使用 Vim 编辑配置文件等）。

第 12 章　Emacs 编辑器

Emacs 是 Editor MACroS（编辑器宏）的缩写，诞生于 1975 年（时间上早于 Vi 编辑器），最初是由 Richard Stallman（理查德・马修・斯托曼）与 Guy Steele 在 MIT（麻省理工学院人工智能实验室）共同编写的。随着 Emacs 的逐步完善，其已成为开源操作系统 Linux 中唯一能与 Vim 编辑器抗衡的文本编辑器。与 Vim 相比，Emacs 一样十分强大，几乎可以完成所有可以想象到的任务。也正因如此，在互联网上甚至引发了 Vim 与 Emacs 之争。

本章将简单介绍 Linux 系统中另一个非常重要的编辑器 Emacs，涉及的知识点如下。

- 介绍 Emacs 编辑器的起源、如何启动和退出编辑器、认识 Emacs 编辑器界面。
- 简单介绍如何在 Emacs 编辑器中打开和保存文件、快速移动光标。
- 介绍如何在 Emacs 中撤销编辑操作、搜索文本、查询并替换文本内容。
- Emacs 编辑器中的常用功能介绍。
- 介绍 Emacs 编辑器中的高级技巧。
- 简单讲解 Emacs 编辑器的目录模式和其他功能。

注意：Emacs 编辑器有一小部分快捷键和命令对大小写敏感，因此使用时需要特别注意区分。

12.1　Emacs 编辑器概述与入门

关于 Emacs 编辑器的作者，大家一定不会陌生。他正是自由软件运动的精神领袖、GNU 计划及自由软件基金会（Free Software Foundation）的创立者，著名的黑客理查德•马修•斯托曼，正是他最早编写了 Emacs。本节将简单介绍 Emacs 编辑器及其入门知识。

12.1.1　Emacs 编辑器概述

20 世纪 70 年代在 MIT 的人工智能实验室，斯托曼见到了在 ITS 上运行的 TECO（Text Editor and Corrector）编辑器，这是一个可以刷新显示的行编辑器。斯托曼对这种所见即所得的文本编辑器产生了非常浓厚的兴趣，后来他又对其进行了重写，并加入了一个新的功能，允许用户重新定义 TECO 的键位。

随着这个新版本在 AI 实验室的流行，许多人为其编写了大量的宏，最后由 Steele 和斯托曼一起将产生的这些宏收集并统一起来，这样就产生了最早的 Emacs。很快这一新的

文本编辑器广为流传，被移植到更多的系统中，并成为这些系统的标准配置。

在其后的一段时间内，许多人开发了应用于不同系统的 Emacs 编辑器，但运行于 UNIX 系统中的 Emacs 却是在 1981 年由 James Gosling 编写的 Gosling Emacs（这个版本在 1984 年成为专有软件）。

1984 年斯托曼提出了 GNU 计划之后，他开始开发 GNU Emacs，这也是 GNU 计划中的第 1 个软件。GNU Emacs 最初是在 Gosling Emacs 的基础之上开发的，与之相同的是，GNU Emacs 也可以在 UNIX 中运行。在随后的 GNU Emacs 版本中，它被赋予更多的功能，结果很快 Gosling Emacs 被 GNU Emacs 替代，成了 UNIX 系统中最为重要的文本编辑器之一。

在 Linux 问世之后，斯托曼将 GNU Emacs 移植到 Linux 系统上。由于 GNU Emacs 整合了一个可以被称为集成化的开发环境，因此 GNU Emacs 在程序员们中间深受爱戴。除此之外，Emacs 还允许用户使用 Lisp 扩展语言对其进行定制，这也是 GNU Emacs 成为全球使用最广泛的文本编辑器的原因之一。

许多用户可能并不关心 GNU Emacs 编辑器有多光辉的过去，而更关心能给自己带来什么样的好处。GNU Emacs 除了能够进行文本编辑之外，还可以做很多事情。

- 利用 Telnet 登录远程主机。
- 收发电子邮件。
- 编写和调试多种编程语言。
- 写日记。
- 作为计算器。
- 浏览网站。
- 玩游戏。
- 煮咖啡。

除此之外还有很多，感兴趣的读者可以阅读相关文档，此处不再一一赘述。

说明： 由于 Emacs 编辑器还存在许多版本，因此在本书中如果没有特别指明，Emacs 就是指 GNU Emacs。

12.1.2　启动 Emacs 编辑器

目前几乎所有的 Linux 发行版都附带安装了 Emacs 编辑器，RHEL5.3 也不例外。但使用 VM Ware Workstation 的读者需要注意，若使用 VM Ware 自动安装系统，则可能会出现没有安装 Emacs 编辑器的情况。如果系统中没有安装 Emacs 编辑器，读者可以从其安装光盘中自行安装。

【命令格式】

与 Vim 编辑器一样，Emacs 也可以直接使用命令启动，基本格式如下：

```
emacs [option] [filename]
```

使用以上命令时，读者可能会发现这样启动的 Emacs 非常慢。这是因为 Emacs 本身是一个非常庞大的软件，并且在启动时 Emacs 还会加载初始化文件~/.emacs。

【常用选项】

如果需要加快 Emacs 编辑器的启动过程，可以在命令后面加上一些选项加速其启动过程，或者重新定义 Emacs 启动后的状态。启动 Emacs 编辑器时，常见的选项及含义如下。

- nw：打开一个文本窗口而非图形窗口。Emacs 启动时，会尝试启动一个图形窗口，这个选项在桌面环境中可以强制打开一个文本窗口（本章中的截图都使用了此选项）。
- q：启动时，放弃加载初始化文件~/.emacs，这个选项对于希望不使用初始化文件的初学者而言非常有用。
- +*n*:*m*：*n* 和 *m* 都是一个数字，Emacs 启动成功后会自动将光标定位于文本的第 *n* 行的第 *m* 个字符处，使用此选项时也可以忽略 *m* 字符选项。这个选项通常用于调试出错的程序源代码。
- u：启动时导入指定用户的初始化文件，这个选项通常用于自定义初始化文件时。
- fn：使用指定的字体作为其窗口字体。

在上面的选项中，需要注意 nw 和 fn 选项并非是由两个短格式的选项组合而成的，它们的长格式分别是 no-winodws 和 font。

除此之外，Emacs 还拥有其他许多选项，可以定义 Emacs 启动后的窗口大小、前景背景颜色、边框和字符颜色等。感兴趣的读者可以参考相关资料，此处不再一一赘述。

【用法示例】

（1）在 X（桌面）环境启动 Emacs 的文本模式，并且放弃加载初始化文件，可以使用以下命令：

```
#使用 emacs 命令强制打开一个文本环境下的 Emacs 编辑器，并且跳过初始化文件
# emacs -nw -q
```

使用以上示例命令时，nw 选项最好单独放在所有选项的最前面，以免引起误解。

（2）如果要在打开 Emacs 编辑器的同时打开文件 example，使用如下命令：

```
# emacs example
```

（3）使用 Emacs 编辑器打开文件 example，并移动光标，使用如下命令：

```
#使用+15:3 表示将光标移动到 15 行的第 3 个字符处
# emacs +15:3 example
```

（4）打开 Emacs 时，可以使用选项 u 指定使用的初始化文件：

```
#使用选项 u 指定使用用户 user 的初始化文件
# emacs -u user
```

除此之外，还可以使用其他的选项启动 Emacs 编辑器，感兴趣的读者可以阅读帮助文件或参考相关文档。

提示：在图形界面中启动 Emacs 时，如果不加任务参数，将会启动 Emacs 的图形界面，由于图形界面中的 Emacs 命令使用方法非常简单，因此本书不再一一赘述。

12.1.3　Emacs 编辑器启动界面

Emacs 编辑器有文本和图形模式两种界面，使用 Emacs 编辑器之前，应该了解其工作

界面。本小节将简单介绍 Emacs 的操作界面。

1．文本模式

如果在字符界面中使用 emacs 命令，系统将会自动启动 Emacs 的文本模式（有时也称为文本窗口）。

使用 Emacs 编辑器新建一个空白文本文件：

```
#使用 emacs 编辑器新建一个名为 example 的文本文件
# emacs example
```

执行以上命令之后，Emacs 启动后会新建一个空白文档，如图 12.1 所示。

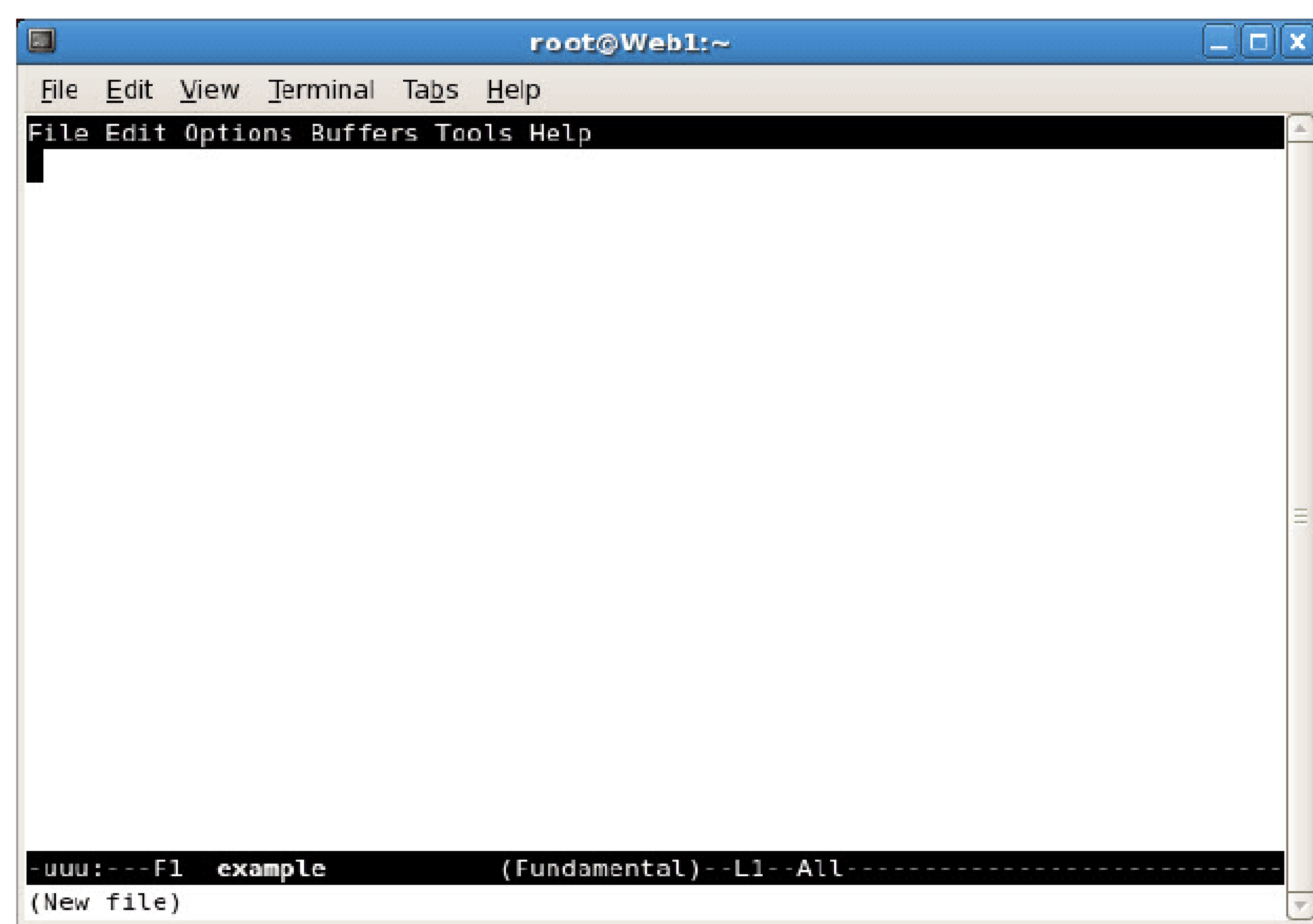


图 12.1　Emacs 编辑器文本模式

从图 12.1 中可以看出，Emacs 编辑器的工作界面明显可以分为 4 个功能不同的部分，从上至下分别是菜单栏、内容显示窗口、模式行（Mode line）、回显区（或者称为小缓冲区）。

- ❑ 菜单栏位于整个编辑器的顶部，其中包括 File（文件）、Edit（编辑）、Options（选项）、Buffers（缓冲区）、Tools（工具）及 Help（帮助）选项。菜单栏中的选项几乎包含了编辑文本时的所有指令。但在文本模式中菜单栏很少被用到，本书将在稍后介绍如何使用菜单栏。
- ❑ 内容显示窗口通常都显示当前正在编辑文本的内容，在此区域内可以添加、修改文本文件的内容。
- ❑ 在内容显示窗口之下，有一个黑底类似于标题栏的模式行（Mode line），主要用于显示当前的编辑状态（因此有时也称为状态栏）。在模式行中通常显示了当前缓冲区名称、缓冲区的内容是否已经被改变等内容。
- ❑ 在模式行之下（即编辑器最后一行）是回显区，有时也常将其称为小缓冲区，小缓冲区主要用来显示消息，输入命令等。

2．图形模式

如果在图形界面中，不使用任何选项直接在终端中使用命令 emacs，系统将会打开 Emacs 的图形模式，如图 12.2 所示。

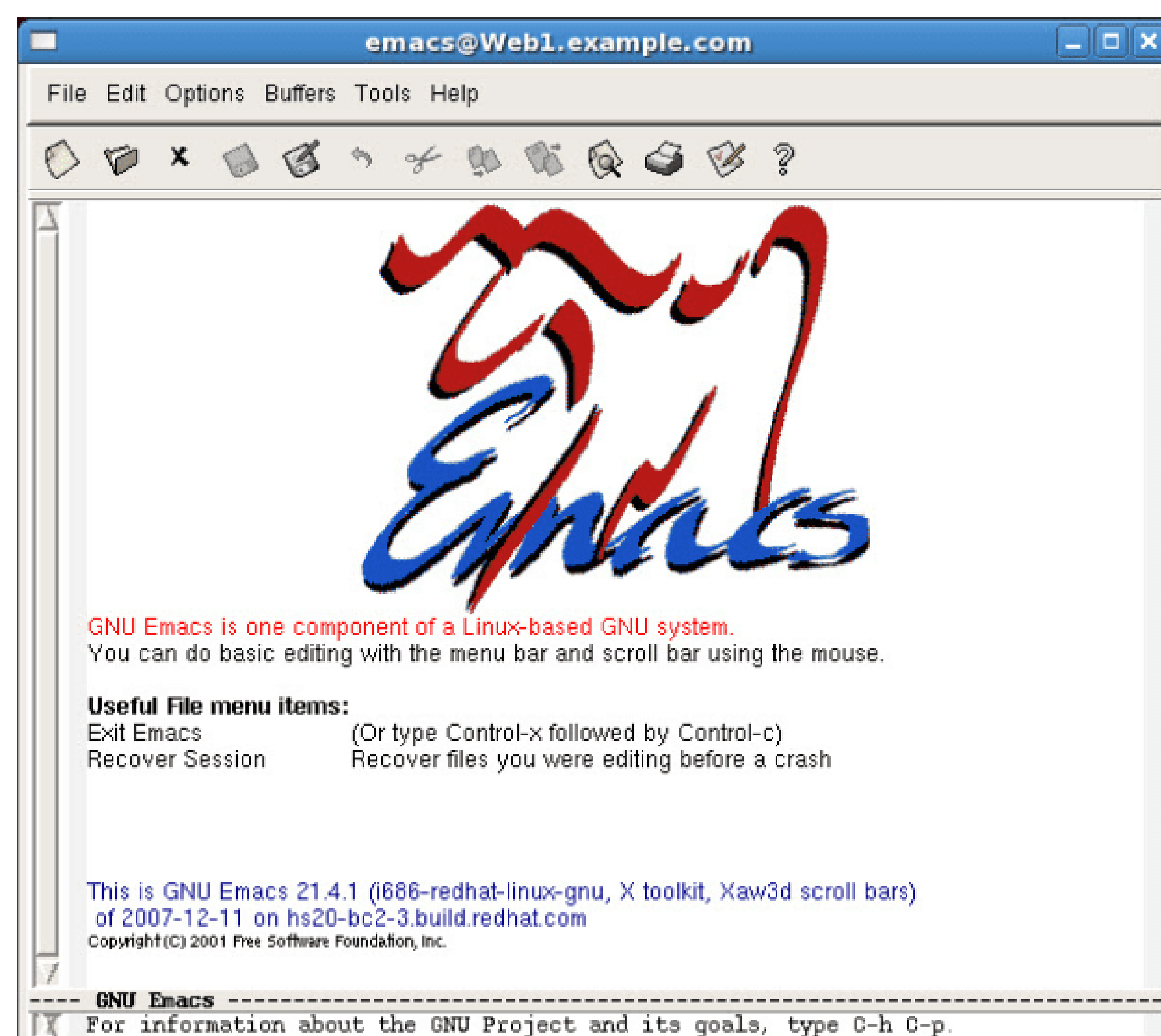


图 12.2　Emacs 编辑器的图形模式

Emacs 编辑器的图形模式使用方法与文本模式类似，因此本书将以文本模式为例讲解如何使用 Emacs，感兴趣的读者可以通过阅读其他文档了解 Emacs 的图形模式。

12.1.4　退出 Emacs 编辑器

退出 Emacs 编辑器可以使用快捷键 Ctrl+X Ctrl+C，即先按住 Ctrl 和 X 键，然后松开 X 键按 C 键，最后松开 Ctrl 键和 C 键。这个快捷键在大多数情况下都可以使用，有些模式中要先使用快捷键 Ctrl+G 取消正在进行的命令。

如果使用此快捷键时，缓冲区内仍有还没有被保存的内容，编辑器会在回显区中提示用户是否需要保存这些内容，如图 12.3 所示。

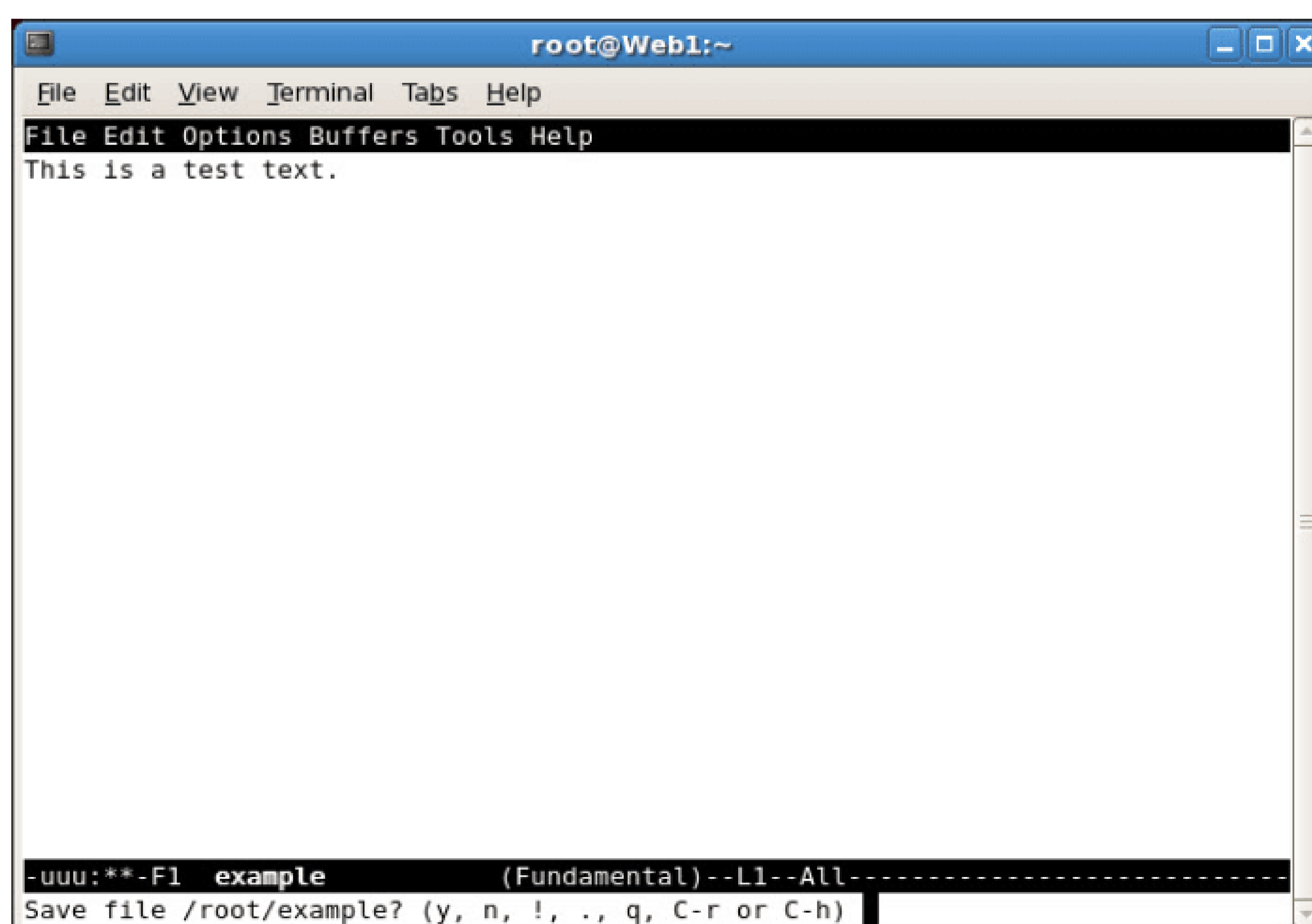


图 12.3　退出 Emacs 时提示用户保存内容

此时按需要按下 y 键保存，或按下 n 键不保存即可退出。

12.2　Emacs 基本操作

在了解了如何启动、退出 Emacs 编辑器，并了解了其基本界面之后，下面介绍如何使用 Emacs 编辑文本。在本节中将简单介绍如何使用 Emacs 的菜单栏、快速移动光标和一些简单的编辑命令。

12.2.1　使用 Emacs 菜单栏

相比 Linux 中的其他文本编辑器而言，Emacs 编辑器的文本模式多了一个菜单栏。对于一些不太熟悉 Emacs 的用户而言，虽然使用起来可能不如图形界面方便，效率也不是很高，但这却避免了在使用初期就去记忆那些繁杂的快捷键。本小节将简单介绍如何使用 Emacs 的菜单栏。

默认情况下，编辑器并没有启动菜单栏，可以理解为：让菜单栏处于可使用状态（MS-DOS 中的应用程序也需要先启动菜单栏，才能使用）。

启动菜单栏可以使用 F10 键，或者快捷键 Esc`（这是一个组合键，先按下 Esc 键，然后按下反引号键`）。菜单栏激活后，Emacs 打开了一个带有选项的缓冲区，如图 12.4 所示。

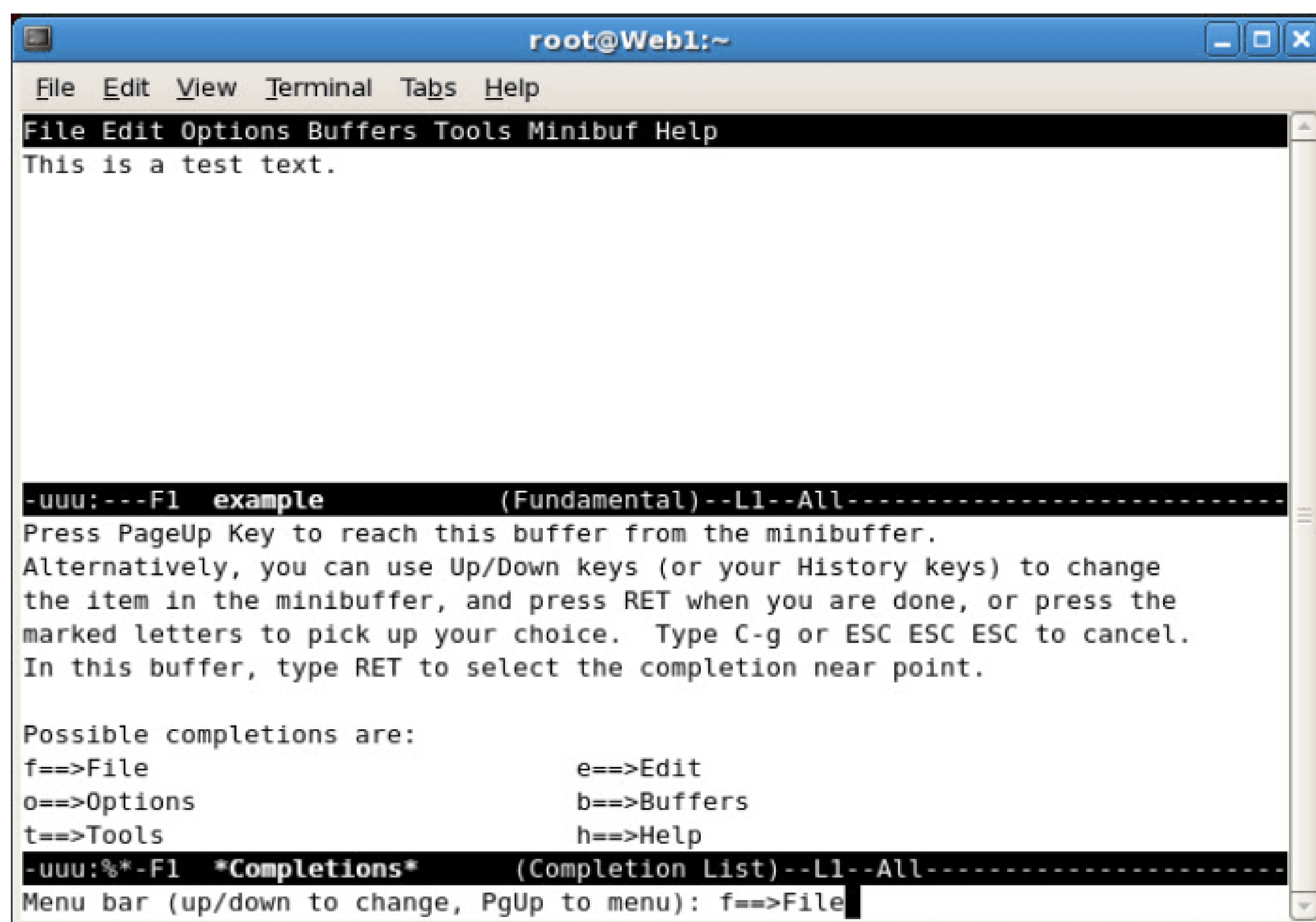


图 12.4　启动菜单栏

弹出选项菜单后，有 3 种方法可选择菜单项。

- ❑ 使用方向键选择菜单项，直到需要的菜单项出现在辅助缓冲区内（即最后一行），然后按下 Enter 键即可。
- ❑ 可以直接输入缓冲区中列出的快捷字母，例如要选择“f==>File”，则按下 F 键即可。
- ❑ 按下 PageUp 键将光标切换到“*Completions*”中，使用方向键将光标移动到需要选择的菜单项上并按 Enter 键。

重复使用上述过程，直到选择到需要的选项。如果在使用过程中需要返回，可以连按 3 次 Esc 键或使用快捷键 Ctrl+G。

注意：由于使用菜单栏十分简便，因此本书中对菜单栏的使用不做过多的讲述，感兴趣的读者可以自行研究和使用。

12.2.2　打开新文件

许多时候需要在已启动的 Emacs 中打开并编辑一个新文件。打开新文件可以使用快捷键和命令两种方式，本小节将简单介绍这两种方式。

1．使用快捷键打开新文件

打开新文件的快捷键为 Ctrl+X Ctrl+F（对应的命令为 find-file），按下快捷键后 Emacs 的界面如图 12.5 所示。

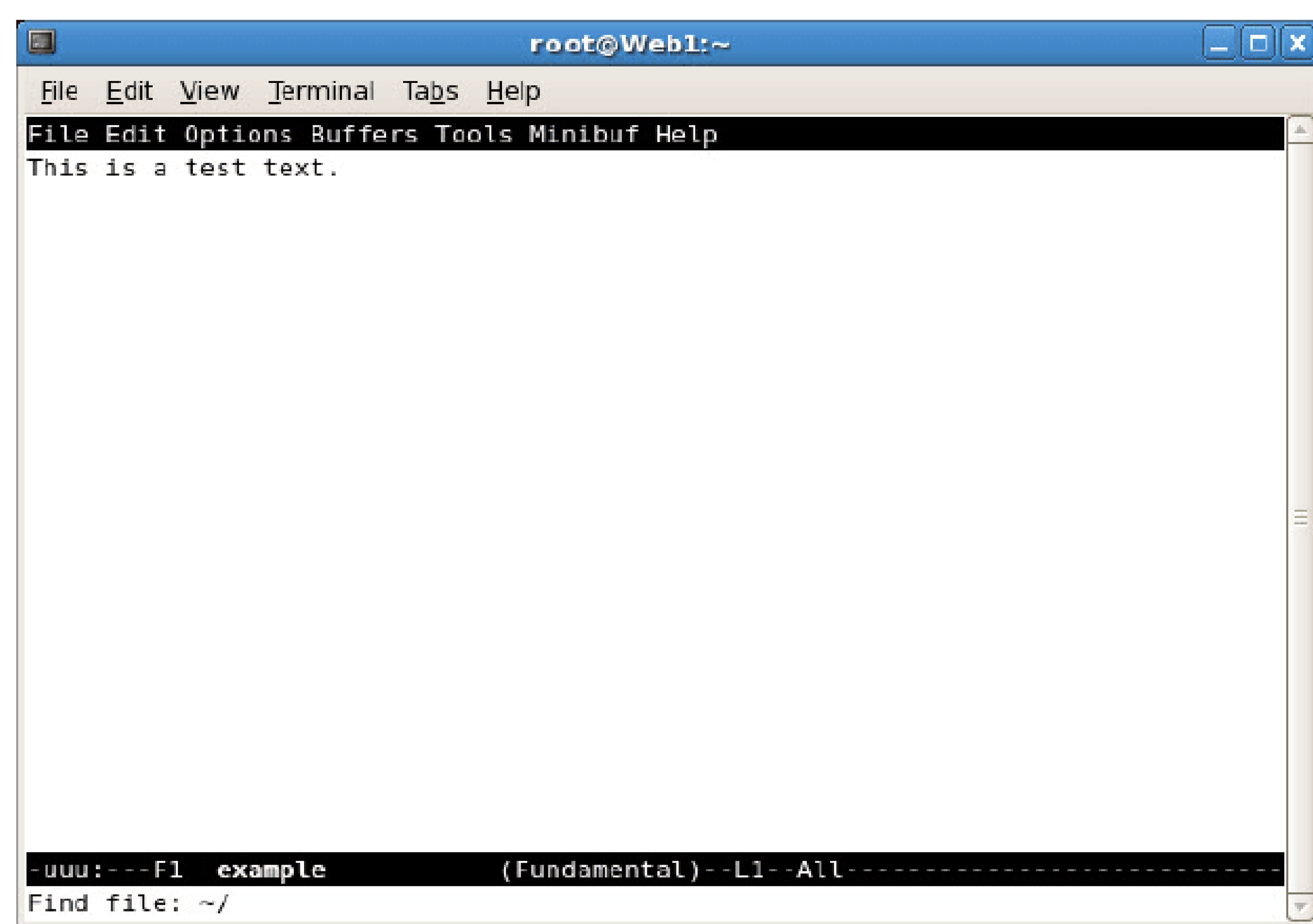


图 12.5　打开新文件

如图 12.5 所示，光标位于小缓冲区内，并且使用当前用户的家目录作为默认路径。此时输入文件位置及名称，按 Enter 键就能打开新文件。

2．使用命令打开新文件

除了使用快捷键打开新文件外，还可以使用命令的方式打开新文件。要在 Emacs 编辑器中输入命令，应该首先按下 Esc 键，然后再按下 X 键。此时无论之前光标的位置在何处，都会跳转到小缓冲区内，最后输入命令 find-file 并按下 Enter 键，即可出现如图 12.5 所示的界面。

提示：大多数快捷操作都拥有一个命令，熟练地使用命令有时可能会事半功倍。

用户在编辑器中输入命令和文件名时，也可以使用命令、文件名自动补全功能。例如在图 12.5 所示界面的小缓冲区内输入/etc/sa，然后按下两次 Tab 键，此时 Emacs 会将/etc 目录中所有以 sa 开头的文件显示到缓冲区上，如图 12.6 所示。

如果输入的目录或文件在目标目录中唯一，则会补全文件名。例如在上面的例子中输入的是/etc/sam，则会将缺失的文件名补全为/etc/samba/。



图 12.6　文件名自动补全功能

3. 替换缓冲区中的文件

有时可能混淆了文件名，打开文件之后才发现打开的文件并不是需要编辑的文件。这时可以使用快捷键 Ctrl+X Ctrl+V（对应的命令为 find-alternate-file），打开正确的文件。这时打开的文件会将当前的文件替换掉，即在当前缓冲区中重新打一个新文件。

替换缓冲区中的文件的方法与打开文件的方法类似，此处不再一一赘述。

12.2.3　保存文件

许多时候需要保存已经编辑好的文件，但保存之后并不退出，可能是还需要编辑其他的文件，也可能是在编辑过程中需要保存等。

（1）保存文件

要保存正在编辑的内容，可使用快捷键 Ctrl+X Ctrl+S。如果当前文件还未命名，Emacs 还会在小缓冲区内提醒用户输入需要保存的路径及文件名。

如果 Emacs 编辑器执行保存命令成功，则会在小缓冲区内提示 Wrote filename，其中 filename 表示文件的名称。如果当前文件没有更新内容，则不会执行保存操作，并在小缓冲区内提示 No changes need to be saved。

（2）将文件另存为

使用快捷键 Ctrl+X Ctrl+S 虽然可以保存文件，但许多时候可能还需要对当前正在编辑的文件执行另存为操作。这时可以使用另一个快捷操作 Ctrl+X Ctrl+W。使用了另存为快捷键之后，Emacs 将会在小缓冲区提示输入路径及文件名称，输入完成之后按 Enter 键即可完成另存为操作。

12.2.4　简单编辑操作

在使用 Emacs 编辑器的过程中，大多数时候都没有使用命令及快捷操作，而是在对文本执行插入、修改、删除等编辑操作。虽然与 Vim 编辑器相比，Emacs 没有过多的模式切

换，但 Emacs 也有独到之处。本小节将简单介绍如何执行最简单的编辑操作。

1．中止正在执行的命令

许多时候可能我们按了一个不知名的组合键，导致光标不在编辑区内（通常都在小缓冲区内）。这时可以使用快捷键 Ctrl+G 或连按 3 次 Esc 键（这两个快捷键的作用是中止当前正在进行的命令）让光标恢复到编辑窗口内。

2．编辑文本

编辑文本的过程非常简单，直接在编辑区中输入要输入的内容即可。除此之外，还需要注意以下内容。

- 如果输入的文本超出了屏幕中一行的长度，Emacs 会在屏幕最右边加上一个反斜杠“\”，并在下一行继续显示输入的内容。不必担心 Emacs 在屏幕最右边加上的反斜杠，Emacs 只是使用这种方式表示这一行的内容没有结束而已。
- 在 Emacs 中最常用的删除键莫过于退格键 Back Space 了。退格键的功能是删除当前光标所在位置的前一个字符，而光标所在位置以后的字符将会替代删除掉的字符，如果多次使用则会删除光标前的所有内容。
- 除了使用退格键外，还可以使用删除键 Delete。删除键可以删除当前光标所在位置的字符。除此之外，Emacs 还定义了一个删除快捷键 Ctrl+D，这个删除快捷键的功能与删除键 Delete 相同。

对于使用 Emacs 的初学者而言，掌握以上内容就足以编辑文本文件了。

12.3　快速移动光标

虽然大多数情况下，都可以通过方向键来快速移动光标，但有时光标移动速度仍然较慢。此时可以配合使用 Emacs 编辑器的快速移动光标快捷键进行定位。本节将简单介绍如何在 Emacs 中快速移动光标。

12.3.1　按字符移动光标

在 Emacs 编辑器中，除了可以使用方向键逐字符移动光标外，还提供了以下几个快捷键。

- Ctrl+B：向前移动光标。
- Ctrl+F：向后移动光标。
- Ctrl+P：向上移动光标。
- Ctrl+N：向下移动光标。

这几个快捷键均是逐字符移动光标，即每按一次就移动一个字符，如图 12.7 所示。

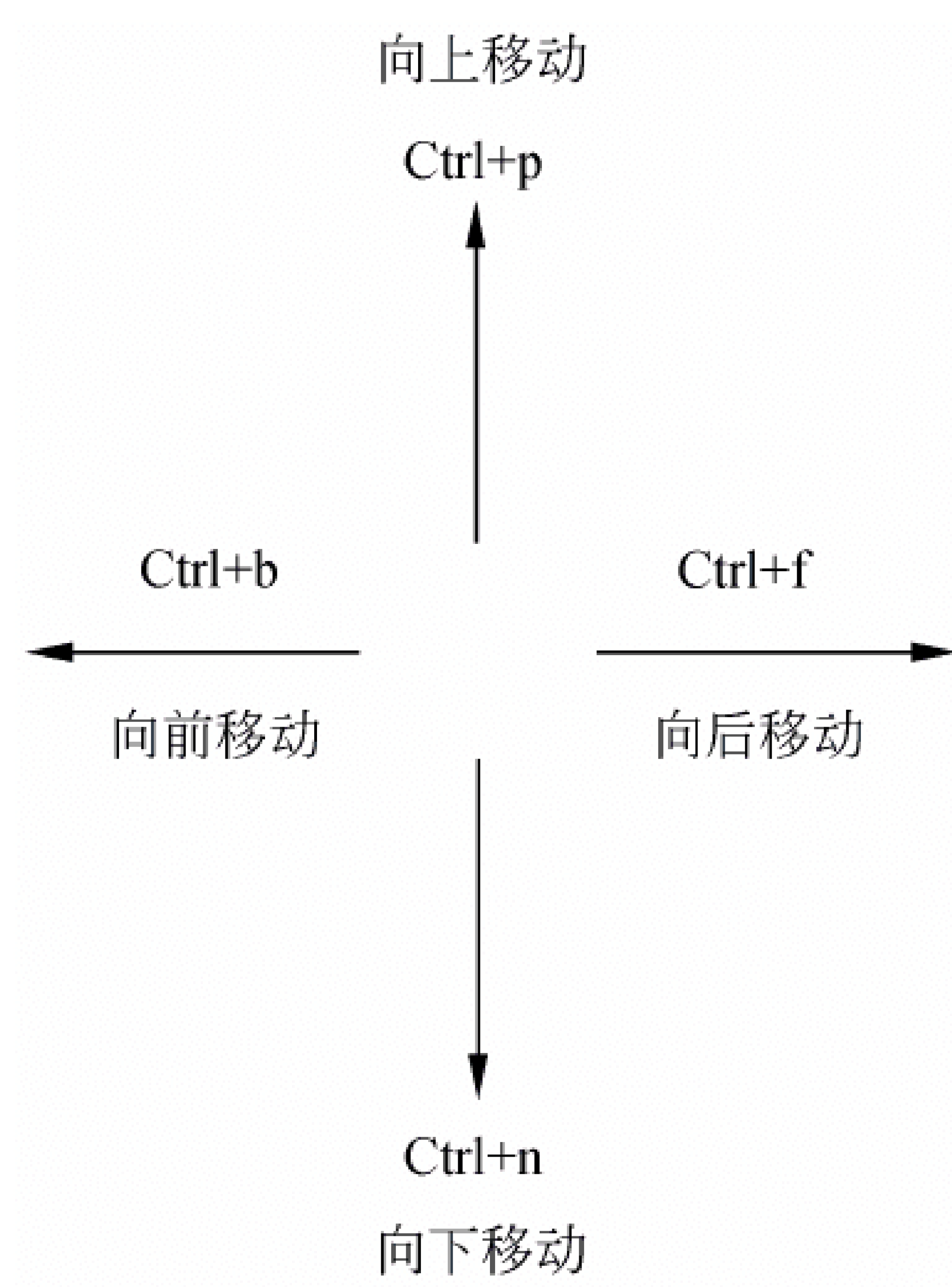


图 12.7　按字符移动光标

如果需要使用这几个快捷键移动多个字符或多行，可

以连续多次使用。

12.3.2　按句子移动光标

在 Emacs 编辑器中，除了较常规的按字符移动光标外，还提供了按句子移动光标的功能。按句子移动光标的快捷键如下。

- ❑ Esc a：将光标前移一句。
- ❑ Esc e：将光标后移一句。

使用上面两个快捷键时，如果当前处于某一句中，将会跳转到当前光标所在句的句首和句尾。第 2 次使用时，才会跳转到上一句的句首、下一句的句尾。Emacs 编辑器可以识别的句分隔符为英文的句号“.”、问号“?”等。

12.3.3　按单词移动光标

在 Emacs 编辑器中，也提供了按单词移动光标的快捷键。

- ❑ Esc b：将光标前移一个单词。
- ❑ Esc l：将光标后移一个单词。

使用这两个快捷键时，Emacs 会分别将光标移动到前一个、后一个单词的词首、词尾。如果需要移动多个单词，可以连续使用多次。

12.3.4　按段落移动光标

Emacs 编辑器还提供了按段落移动光标的方法。

- ❑ Esc Shift+[：即 Esc {，将光标前移一个段落。
- ❑ Esc Shift+]：即 Esc }，将光标后移一个段落。

通常 Emacs 判断段落的标准是换行符，因此移动光标时，会将光标移动到换行符之后或换行符之前。

使用上面这两个快捷键时，如果光标处于段落中，Emacs 编辑器会将光标移动到段首、段尾。

12.3.5　滚动屏幕

与 Vim 编辑器一样，Emacs 编辑器也没有滚动条，因此只能使用快捷键的方式滚动屏幕。在 Emacs 编辑器中，除了可以使用编辑键 Page Up、Page down 之外，还可以使用以下快捷键。

- ❑ Ctrl+V：显示下一屏。
- ❑ Esc v：显示上一屏。
- ❑ Esc >：快速移动到文本结尾处。
- ❑ Esc <：快速移动到文本起始处。

使用以上快捷键，可以快速滚动屏幕，并将光标移动到想要编辑的位置上。

🔔**技巧：**如果觉得重复执行某些快捷键比较麻烦，可以使用 Ctrl+u *n*（*n* 为重复次数，默认为 4 次）和 Esc *n* 重复执行 *n* 次后续快捷键、命令。

12.3.6　其他移动光标的技巧

在前面几个小节中，简单介绍了 Emacs 编辑器中移动光标的常用技巧。在本小节中将继续介绍几个更简单的移动光标的小技巧。

1．按行移动光标

在 Emacs 编辑器中，除了使用编辑键 Home、End 将光标移动到行首、行尾外，还可以使用以下快捷键。

- Ctrl+A：将光标移动到行首。
- Ctrl+E：将光标移动到行尾。

需要注意的是，这两个命令只能在当前行使用。无论使用多少次，都无法越过行首、行尾，将光标移动到上一行、下一行。

2．将当前行移动到屏幕中间

有时编辑位置处于屏幕上方、下方，这时可能会比较麻烦，因为无法完全看到编辑位置附近的文本。特别是在编辑配置文件时，无法看到附近的注释就可能导致输入错误。这时可以使用快捷键 Ctrl+L（字母 l），将当前光标的位置移动到屏幕中间。

需要注意的是，使用这个快捷键需要保证文本处于屏幕中间时，上下文的文本足够“铺满”整个屏幕。

3．将光标移动到指定的行

使用 Emacs 编辑器编写代码时，经常需要快速移动到指定的行。这是因为调试代码时，通常都提示错误出现在某一行附近。

要将光标移动到指定的行，可以先使用快捷键 Esc x，然后输入命令 goto-line，并按下 Enter 键。此时 Emacs 编辑器将提示用户输入行号，如图 12.8 所示。

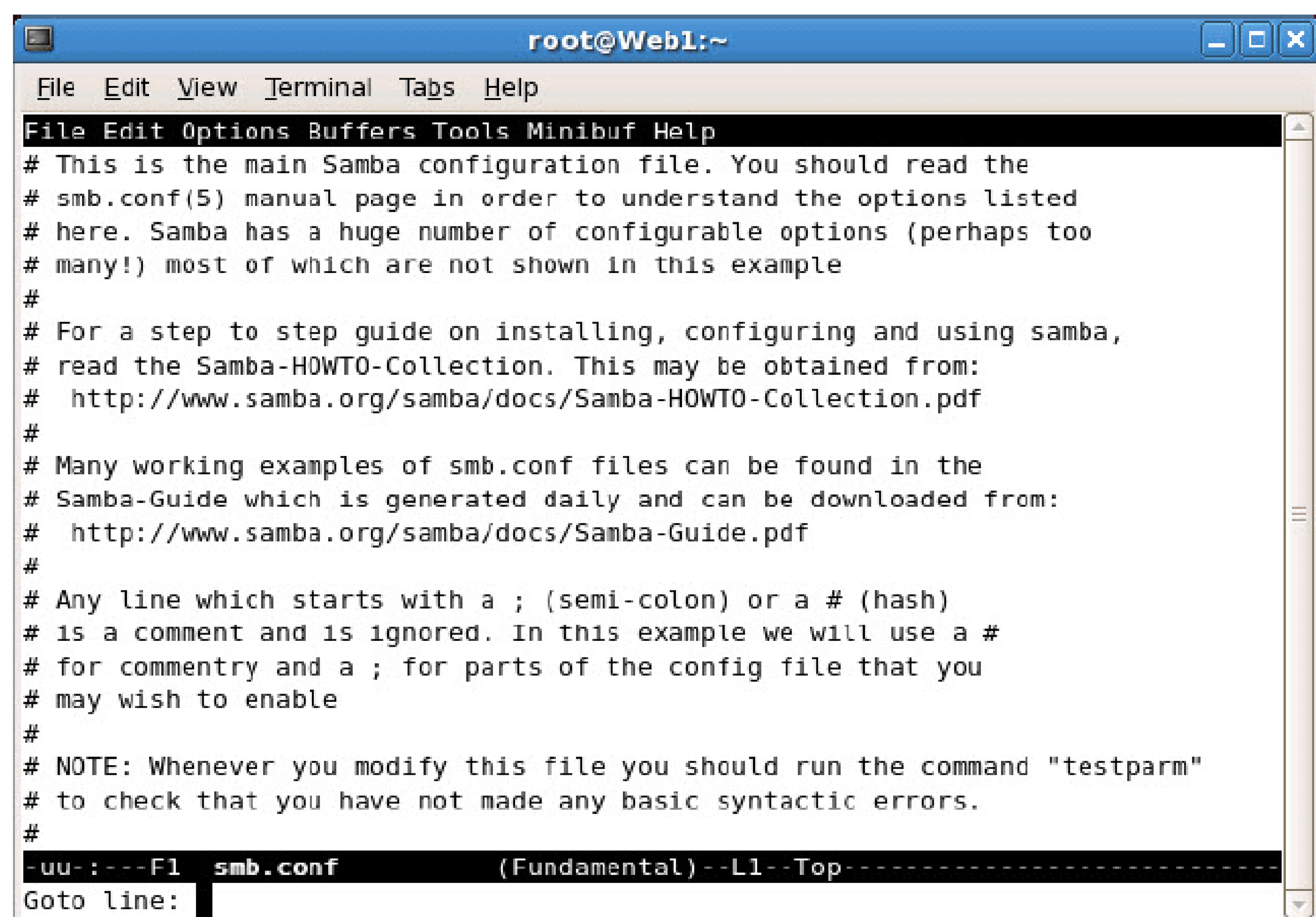


图 12.8　提示用户输入行号

在图 12.8 中，用户可以在小缓冲区输入行号，按下 Enter 键即可跳转到指定的行。

4．Emacs编辑器中的移动光标快捷键

在本节中介绍了许多移动光标的快捷键，这些快捷键的功能示意图如图 12.9 所示。

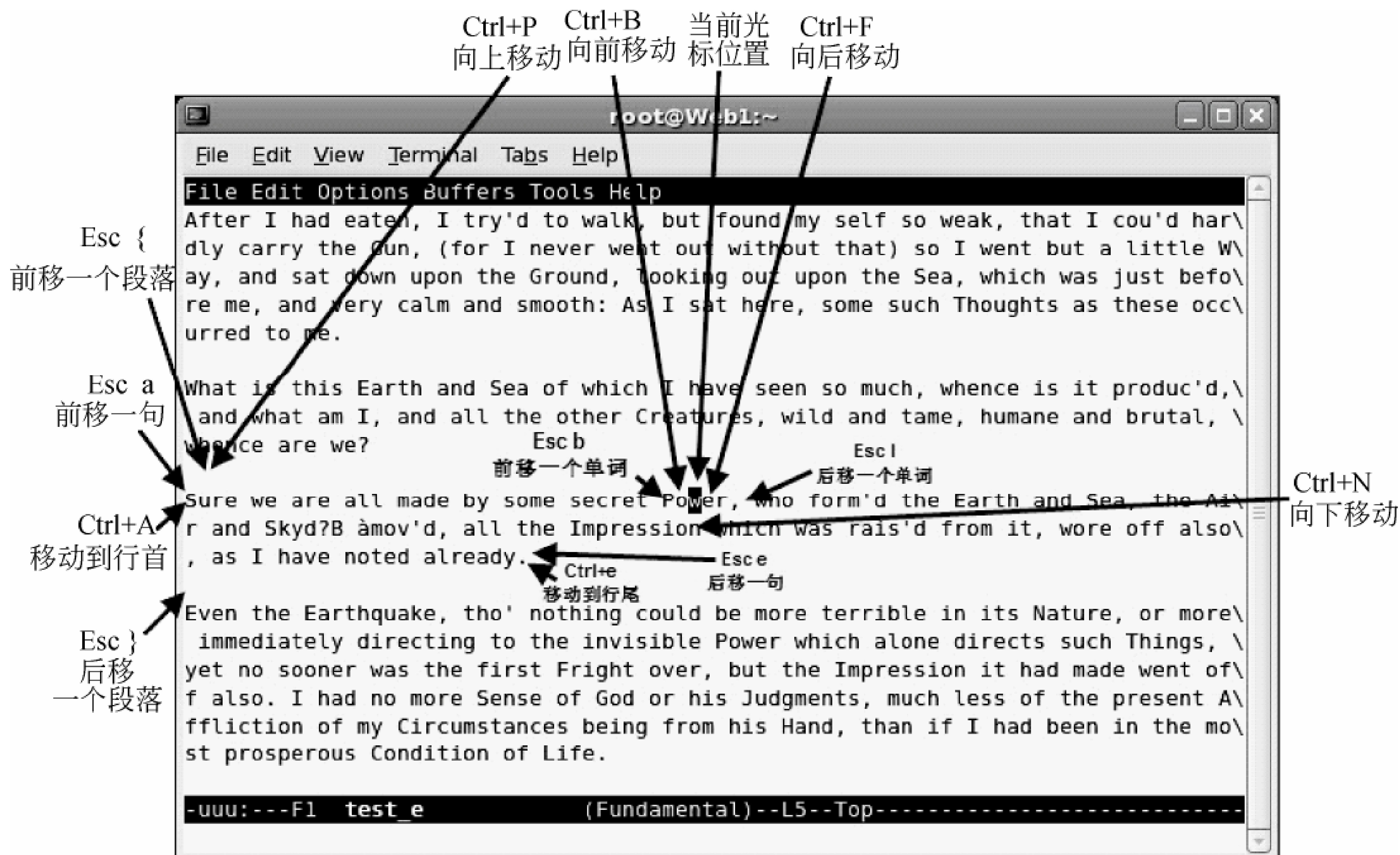


图 12.9　移动光标的快捷键

虽然 Emacs 编辑器中附带了许多可以移动光标的快捷键，但建议初学者养成自己的习惯即可，不必过于追求掌握每种用法。

12.4　Emacs 编辑器的常用功能

在本章的前面几节中，介绍了 Emacs 编辑器的基本情况、使用方法和技巧等内容。这只是文本编辑器的最基本操作，对于整个 Emacs 编辑器而言，这些仅仅是九牛一毛，但就是这样也足以完成大多数文本编辑任务了，由此可见 Emacs 功能的强大。在本节中将介绍 Emacs 编辑器的常见功能。

12.4.1　撤销与恢复

许多时候要对一个已经存在的文本进行编辑，可能是插入新的内容、修改原有的内容及删除不合适的内容等。然而有时会发现过去执行的某一个编辑操作可能并不适当，或已经修改过的内容并不理想，这时可能需要撤销已经执行过的编辑操作，恢复原有文本。

（1）撤销修改

默认情况下，Emacs 编辑器将会记录一些文本修改的记录，可以使用这些记录来还原

文本的原有内容。

撤销操作的快捷键是 Ctrl+Shift+-（即 Ctrl+_），使用此快捷键可以快速撤销最后一次操作。如果多次使用，Emacs 最终会撤销所有编辑操作。

（2）恢复修改

许多时候在使用撤销快捷键之后，可能在仔细查看之下又觉得最后这个操作确实非常有必要，这时可以恢复最后撤销修改的内容。

要恢复修改，可以使用快捷键 Ctrl+F Ctrl+Shift+-。需要注意的是这个快捷操作键只在撤销快捷键前面加上了一个 Ctrl+F。使用时先按下前面的 Ctrl+F，然后可以连续按多次 Ctrl+Shift+-，以恢复多个撤销修改的内容。

注意：Emacs 编辑器用于记录编辑操作的缓冲区有一定上限，因此如果在编辑过程中进行了许多操作（多到 Emacs 不可能完全记录所有的操作时），使用撤销键就无法恢复文本的内容至初始状态。

12.4.2　搜索功能

在 Emacs 编辑器中，文本的搜索功能分为两种模式，其一是输入需要查找的文本内容，然后进行查找，这种模式在 Emacs 中称为增量查找。其二是一边输入文本内容，一边查找，这种查找模式在 Emacs 中称为非增量查找。本小节将简单介绍如何使用增量查找和非增量查找。

1．增量查找

增量查找是最常用的查找方式，从用户输入第 1 个字符开始，Emacs 即开始对文本进行查找，并为查找到的内容添加上颜色，以区别于其他文本内容。随着用户不断地输入文本，Emacs 的查找变得越来越精确，最后找到要查找的文本。如果用户输入的文本没找到，Emacs 将会显示相关错误消息。

要进行增量查找，可以使用快捷键 Ctrl+S 和 Ctrl+R，按向下、向上两个不同的方向查找。

例如按下快捷键 Ctrl+S 进行向下的增量查找，此时光标会自动跳转到小缓冲区内。接着输入文本 th，此时 Emacs 会将查找到的 th 使用不同的颜色标识出来，如图 12.10 所示。

在图 12.10 中，Emacs 将查找到的文本使用蓝色标识出来，当前光标位置的文本使用粉红色标识出来。

查找到文本内容时，用户可以执行的操作如下。

- 按下 Enter 键，将光标跳转到当前文本处。
- 如果需要在查找过程中退出查找，则可以连按两次 Enter 键或使用 Ctrl+G 键返回。
- 如果要查看下一个、上一个找到的文本，可以使用快捷键 Ctrl+S 和 Ctrl+R 快速切换，直到找到需要的内容。

2．非增量查找

虽然增量查找有时显得非常方便，但是有时可能会觉得使用增量模式进行查找的过程

非常繁琐，这时就可以使用非增量查找。使用非增量查找时，Emacs 编辑器会等待用户将要查找的文本输入完成之后才开始查找。如果找到，则将光标移动到目标文本之后，如果未找到，则给出相关提示。

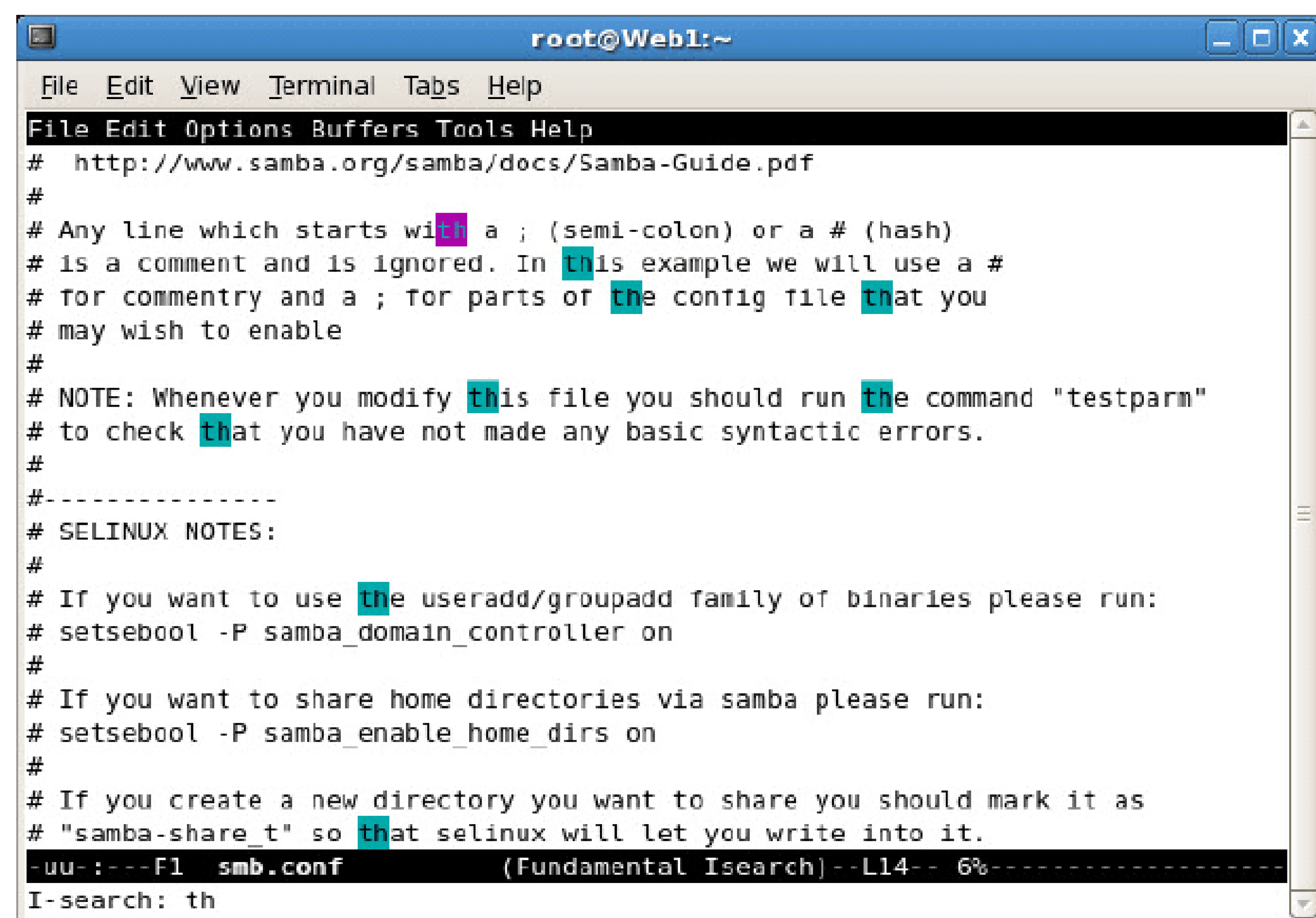


图 12.10　增量搜索

要使用非增量查找，可以使用快捷键 Ctrl+S Enter 和 Ctrl+R Enter，与增量查找相同，这两个快捷键也是向下和向上查找。与增量查找不同的是，这两个快捷键在增量查找快捷键的最后添加了一个 Enter 键。

非增量查找的使用方法与增量查找基本相同，此处不再一一赘述。

注意：与 Vim 不同，在 Emacs 编辑器中，无论使用何种模式，向上和向下查找都不会越过文件头或文件尾。

12.4.3　查找并替换

在编辑文本的过程中，许多时候都需要批量性地处理一些错误。例如在一个文档中将某个单词拼错，或将某个单词替换为另一个单词等。如果人为地来处理这些错误，将是一件非常繁琐的事情，并且可能做得并不好，此时可以使用 Emacs 自带的查找替换功能。

（1）要使用查找并替换功能，可以先执行命令 replace-string，然后在小缓冲区中输入需要查找的文本内容和替换后的文本内容，即可执行替换操作。此时 Emacs 会从光标的当前位置开始查找，将所有找到的所有文本进行替换，最后在小缓冲区内提示一共执行替换的次数。

（2）使用上述方法进行查找替换时，如果替换的文本内容很少，可能没有太大的问题。但如果文本内容较多，且都不加区别地替换，可能是不适当的。这时需要一种更加安全的查找替换方式，即每查找到一处文本，都询问用户是否需要将其替换。

如果要使用比较安全的查找替换功能（通常称为查询替换），可以先按下 Esc %键（即 Esc Shift+5 键），然后输入要查询的文本和替换的文本后按 Enter 键。

使用查询替换时，Emacs 会将所有查找到的文本进行一一询问之后，再执行替换操作，如图 12.11 所示。

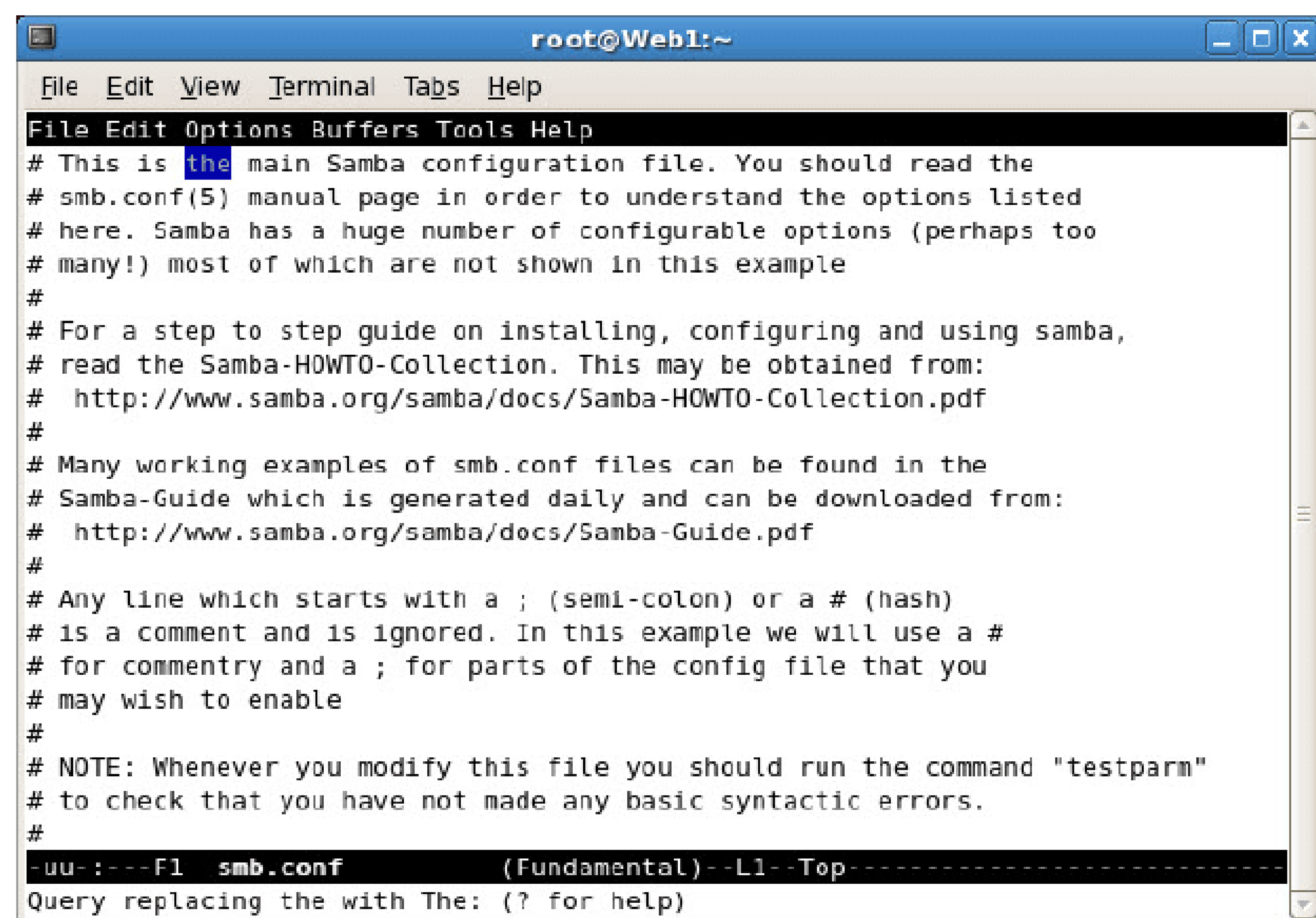


图 12.11　使用询问替换功能

在图 12.11 中，Emacs 编辑器查找到文本（以蓝色标识），并询问用户应该如何处理当前文本。这时可以采取的操作如下。

- Space（空格）或者 y：替换当前文本并查找下一个。
- n：不替换当前文本，直接查找下一个文本目标。
- .：句点号表示替换并退出。
- ,：逗点号表示替换，并显示已替换情况，继续查找可以使用空格或 y。
- !：替换所有查找到的文本并且不再询问。
- ^：返回上一次替换的位置。
- Enter（回车）或 q：退出查询替换。
- Ctrl+R：进入递归编辑状态。
- Esc Ctrl+C：退出递归编辑状态。

查询替换时，有时会发现某一处文本需要修改，这时要对其进行修改，就必须要停下正在进行的替换工作。如果放弃修改，可能替换工作完成之后又无法找到此位置。Emacs 编辑器的开发人员也想到了这一点，遇到这种情况时，可以使用快捷键 Ctrl+R 进入递归编辑状态。在递归编辑状态中可以编辑文本内容，编辑完成后又可以使用快捷键 Esc Ctrl+C 退出递归编辑状态，并查询替换。

注意： 无论使用何种查找替换功能，Emacs 编辑器总是从当前光标位置开始查找并替换，并且这个过程不会越过文件尾。因此如果需要对全文进行查找替换，可以事先将光标停留在文件首或文件尾。

12.4.4　复制、剪切和粘贴

编辑文本时，可能需要经常复制、剪切文本，然后在文本的另一位置粘贴这些文本。Emacs 编辑器中提供了比较复杂的复制、粘贴的操作（复制、粘贴操作涉及删除环，这将在下一节中介绍），但对于用户而言，无须了解这么多。本小节将简单介绍 Emacs 编辑器中最简单的复制、剪切、粘贴操作。

（1）复制

在进行复制、粘贴之前，需要选择要复制的文本。选择文本的方法是将光标移动到复制文本的开头，使用快捷键 Ctrl+Space（即 Ctrl+空格）。按下快捷键之后将在小缓冲区内显示 Mark set（设置标记点），如图 12.12 所示。

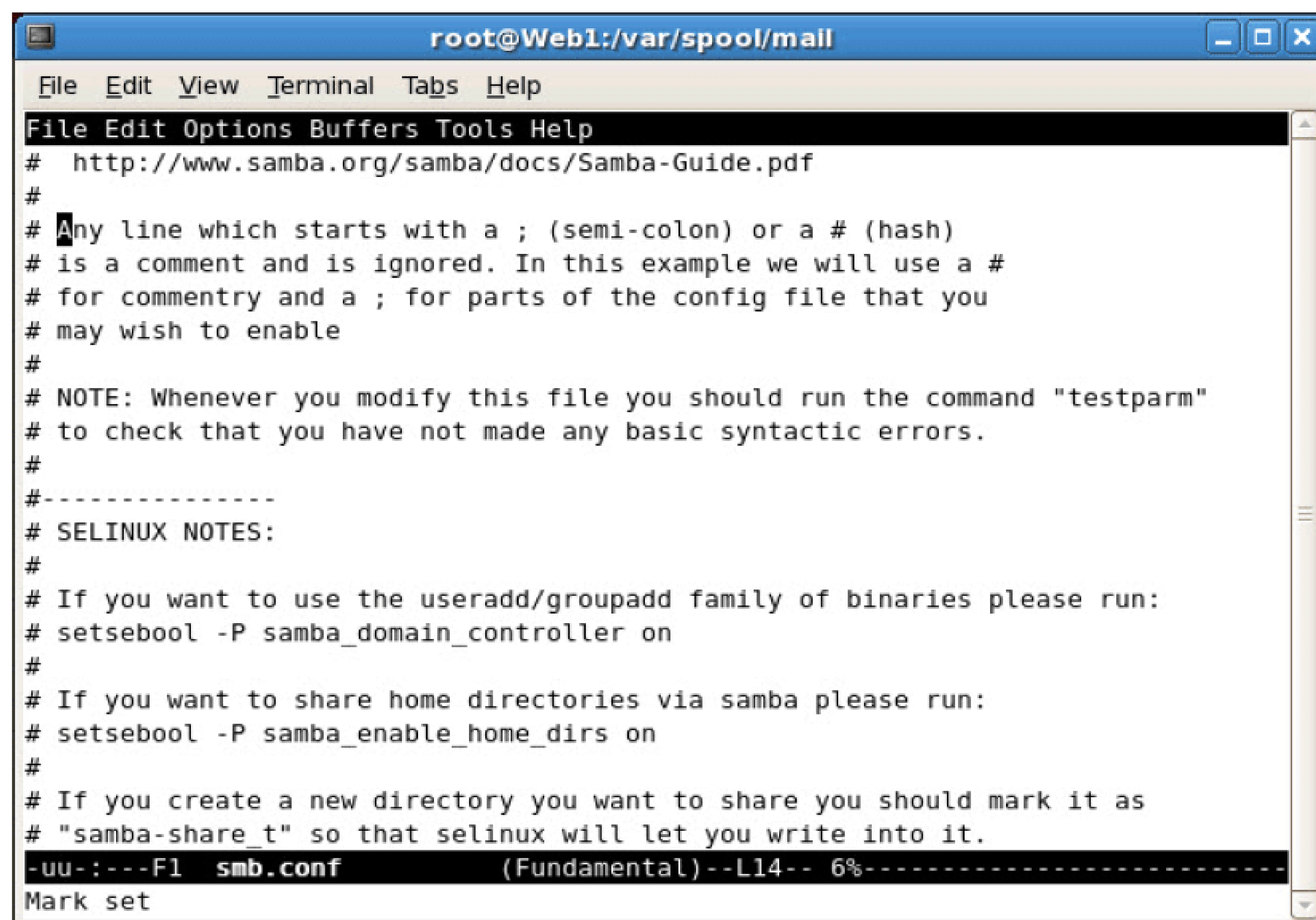


图 12.12　设置开始标记

当小缓冲区内出现如图 12.12 所示的 Mark set 标记后，就可以将光标移动到文本尾，按下快捷键 Alt+W（或 Esc W），此时 Emacs 编程器中的光标将会跳转至标记点（要复制的文本头）。快速闪烁之后，光标将回到要复制的文本尾，此时要复制的文本就已经复制到缓冲区内了。

（2）剪切

剪切操作与复制操作类似，先将光标移动到要剪切的文本头，使用快捷键 Ctrl+Space 设置标记点。然后在需要复制的文本尾按下快捷键 Ctrl+W，即可将文本剪切到缓冲区中。

（3）粘贴

执行复制、剪切之后，就可以将光标移动到需要粘贴的位置，并按下快捷键 Ctrl+Y 完成粘贴操作。

提示：在某些终端中（主要是 Windows 或一些图形界面下的虚拟终端）快捷键 Ctrl+Space 用于切换输入法，此时可以使用快捷键 Ctrl+Shift+2（即 Ctrl+@）替代。

12.5　Emacs 编辑器的高级技巧

如果仅仅是用 Emacs 编辑器编辑配置文件，那么前面几节中介绍的内容就已经足够了。但 Emacs 编辑器的功能远不止于此，还有许多功能。在本节中，将介绍 Emacs 编辑器的几个高级技巧。

12.5.1　删除环

有时可能希望能够让复制、剪切和粘贴功能执行得更自由，例如需要对多个处于不同位置的文本执行复制、剪切和粘贴。这时就需要用到删除环功能。

当使用某些特殊的删除功能键时，Emacs 编辑器会将删除（也可用于复制）的文本放入一个被称为删除环的缓冲区中。然后将光标移动到需要粘贴的位置，按下粘贴快捷键 Ctrl+Y 即可完成粘贴。与前面介绍的复制、剪切不同，删除环中可以保存多次操作的结果。

（1）例如需要使用删除环剪切图 12.13 所示文本。

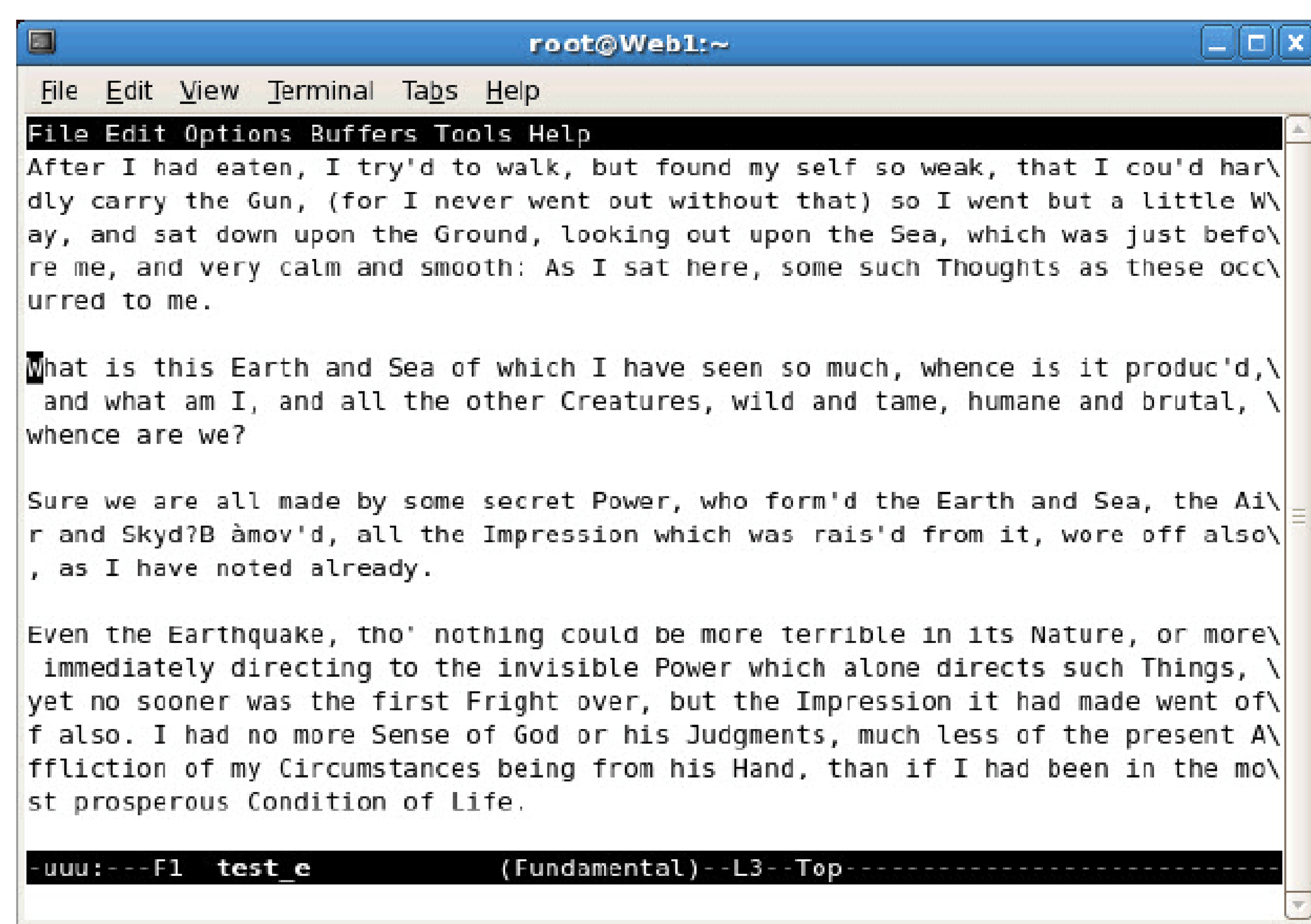


图 12.13　删除环的示例文本

在图 12.13 中，已经将光标移动到第 2 行的行首，然后执行删除行快捷键 Ctrl+K4 次。这个操作会删除当前光标所在行，以及以“Sure”开头的行（包括其中的空行）。

（2）然后将光标移动到图 12.13 中所示文本的最后一个空行上，即以“Even”开头的段落后面的空行。

（3）执行 1 次粘贴快捷键 Ctrl+Y，前面删除的两个段落就已经粘贴到空行上了，如图 12.14 所示。

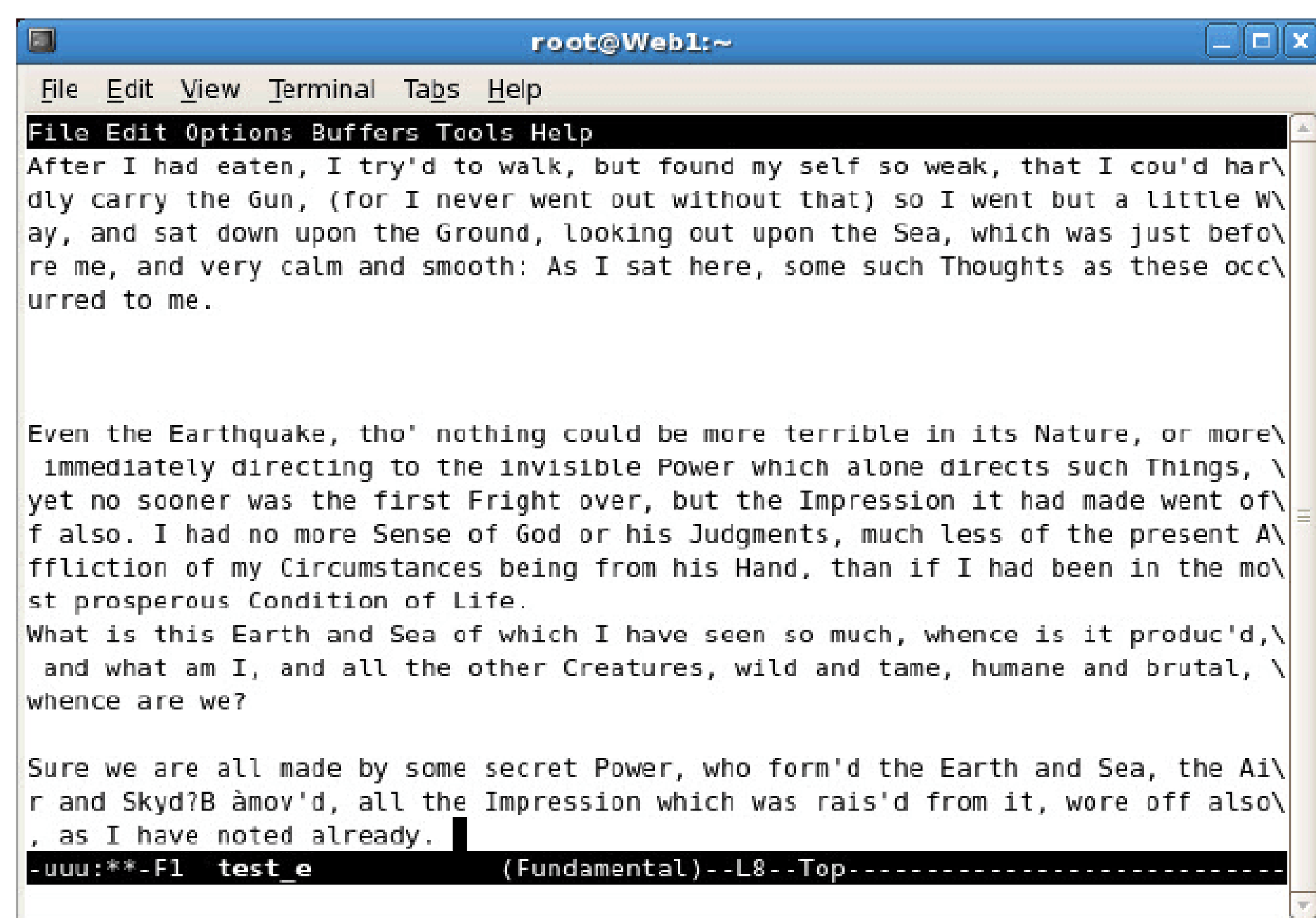


图 12.14　粘贴删除环中的文本

在图 12.14 中可以看到，只使用 1 次粘贴快捷键就将删除了 4 次的内容粘贴到目的位置了。当然如果需要，还可以粘贴更多内容。

从上面这个示例中可以看出，删除环可以将多次删除的内容一次粘贴到目标位置，有时会非常方便。

【哪些操作会被放入删除环】

所有的删除都会放入删除环？当然不是，只有一些特定的快捷操作会放入删除环。常见的操作如下。

- 使用 Ctrl+K 删除的所有内容都会放入删除环（这个快捷键的功能是删除当前光标至行尾的所有内容）。
- 使用 Ctr+W 删除的所有内容将放入删除环（这个快捷键的功能是删除文本块）。
- 使用 Esc w 和 Alt+W 复制的所有内容都会放入删除环。
- Esc d 删除的所有内容（这个快捷键的功能是删除光标处的单词）。

将文本放入删除环时需要注意的是，放入删除环的操作应该连续进行。例如先使用 Ctrl+K 删除几行内容，然后使用 Esc d 删除几个单词等，执行粘贴时 Emacs 会按删除的顺序进行粘贴。

如果使用了几个可以放入删除环的操作，然后再使用 Ctrl+D、Delete 等删除字符就会打断放入删除环操作。Emacs 会清除删除环中的所有内容，以便重新记录。

技巧：不要尝试向删除环中放入过多的内容，否则容易产生一些混乱，并且可能会出现删除无法保存更多内容（默认为最近 30 次），从而导致数据丢失的现象。

12.5.2　编辑文本区域

在前面几节的介绍中，Emacs 总是按行进行编辑操作的，有时可能会觉得这不太方便。例如要删除多行内容时，不得不逐行进行删除；删除一行中的某一段时，不得不逐字符进行删除。这时就可以使用文本区域（有时也称为文本块）执行编辑操作。

文本区域在上一节中的复制、剪切中已经有所介绍，此处简单介绍一下如何选取、编辑文本区域。

（1）先将光标移动到文本区域的开头，此时可以按下 Ctrl+@键（即 Ctrl+Shift+2）。此时编辑器的小缓冲区内将显示 Mark set。

（2）完成上述操作后，就可以将光标移动到文本区域的结束处。为了验证文本区域的选择是否正确，可以按快捷键 Ctrl+X。使用此快捷键后，光标快速闪烁至文本区域开头，然后返回结束处。用户可以通过观察验证文本区域是否已正确选择。

（3）确认正确选择文本区域后，就可以使用复制、删除等操作了。

12.5.3　书签功能

在较长的文本中往复修改、编辑是一件比较费时的事，因为需要在多个编辑点之间来回切换，可能会耗费较多的时间。此时，可以利用 Emacs 编辑器自带的书签功能。使用

Emacs 编辑器的书签功能，不仅可以快速将光标移动到要编辑的位置，还可以将光标移动到另一个文件的编辑位置。本小节将简单介绍 Emacs 编辑器的书签功能。

1．设置书签

设置文本文件的书签时，可以使用快捷签 Ctrl+x r m（也可以使用命令 bookmark-set），之后光标会跳转到小缓冲区中，如图 12.15 所示。

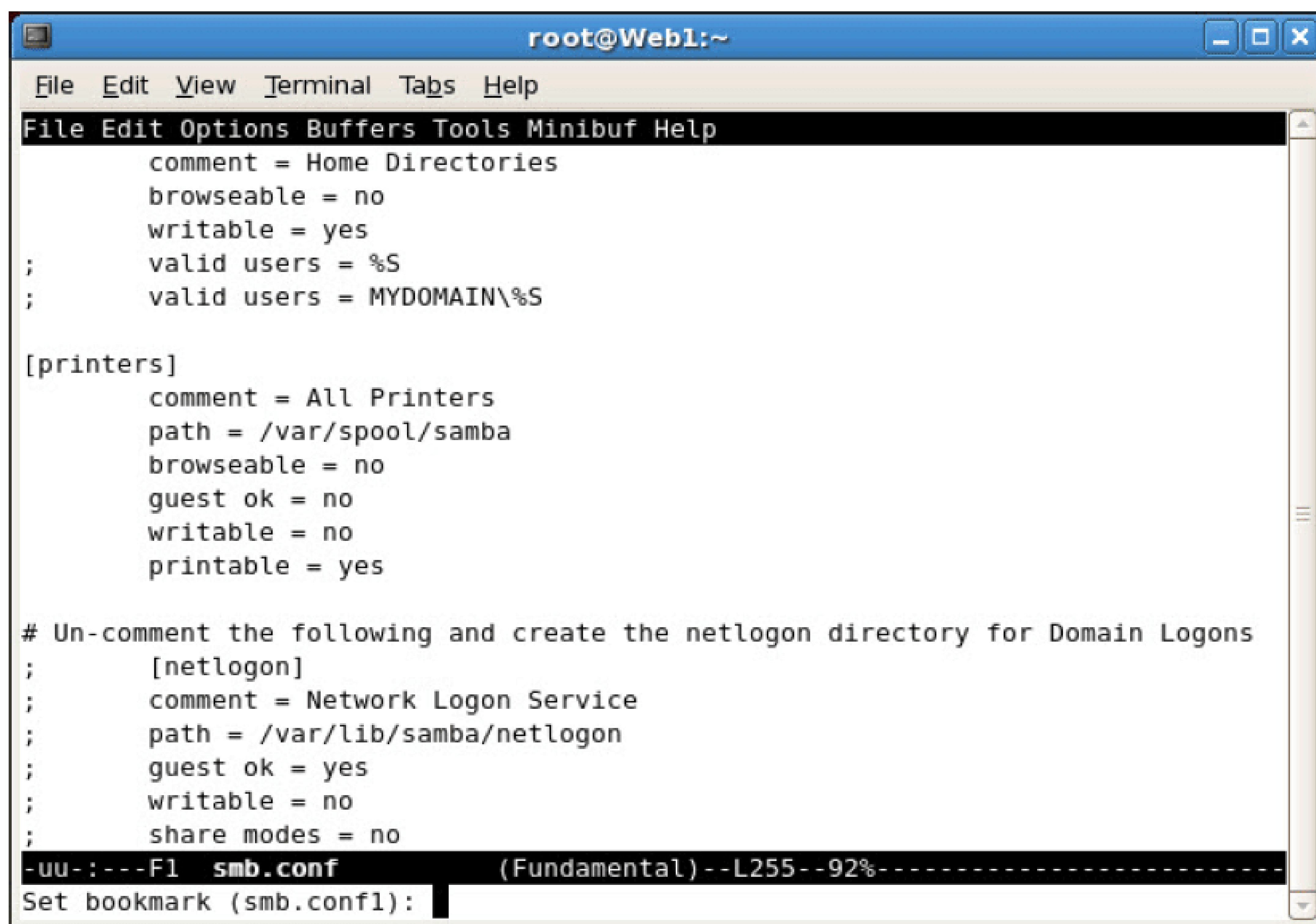


图 12.15　设置书签

在图 12.15 中，Emacs 编辑器在小缓冲区内提示用户输入书签名称，默认使用当前文件名或已存在的书签作为名称。用户只需要输入书签名并按下 Enter 键就可以完成设置。

设置书签时需要注意以下几点。

- ❑ 虽然 Emacs 编辑器没有对书签名称作限制，但应该尽量避免使用特殊字符作为名称或名称的一部分。
- ❑ Emacs 没有对书签名的长度作限制，但应该尽量使用简短并且容易记忆的字符作为名称。
- ❑ 如果使用已经存在的书签名，Emacs 会更新书签的位置，而不管它是否为另一个文件的书签。因此设置书签名时，需要保证书签名独一无二。

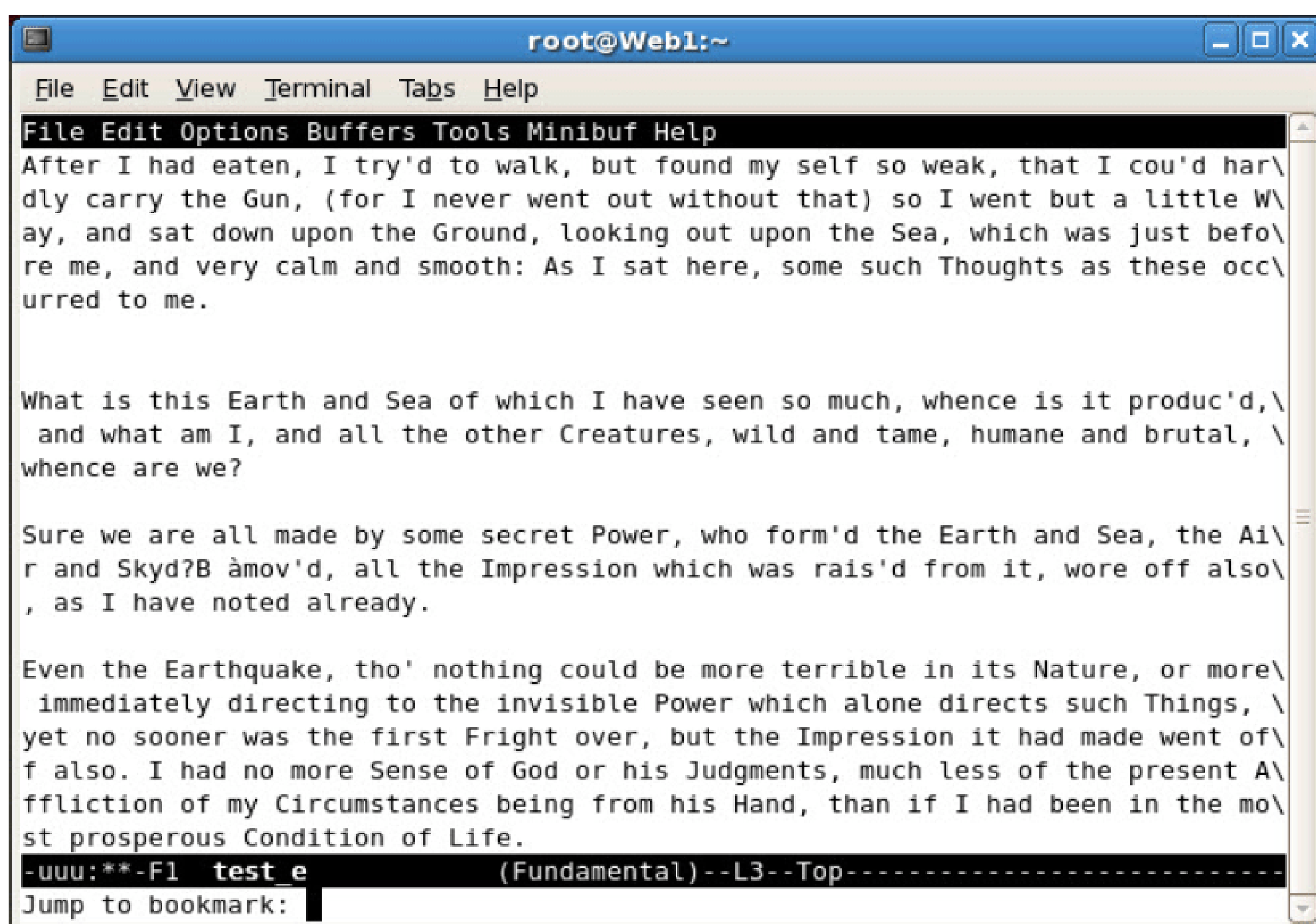
设置书签后，Emacs 会将书签设置保存到文件~/.emacs.bmk 中，因此不要删除这个文件，以防止书签设置丢失。

2．跳转到书签

要跳转到书签时，可以使用快捷键 Ctrl+x r b，Emacs 会将光标移动到小缓冲区中，如图 12.16 所示。

在图 12.16 中，用户只需要输入书签名就可以跳转到指定的书签位置。

如果用户忘记书签名，可以使用补全功能，按两次 Tab 键，查看书签名列表，如图 12.17 所示。



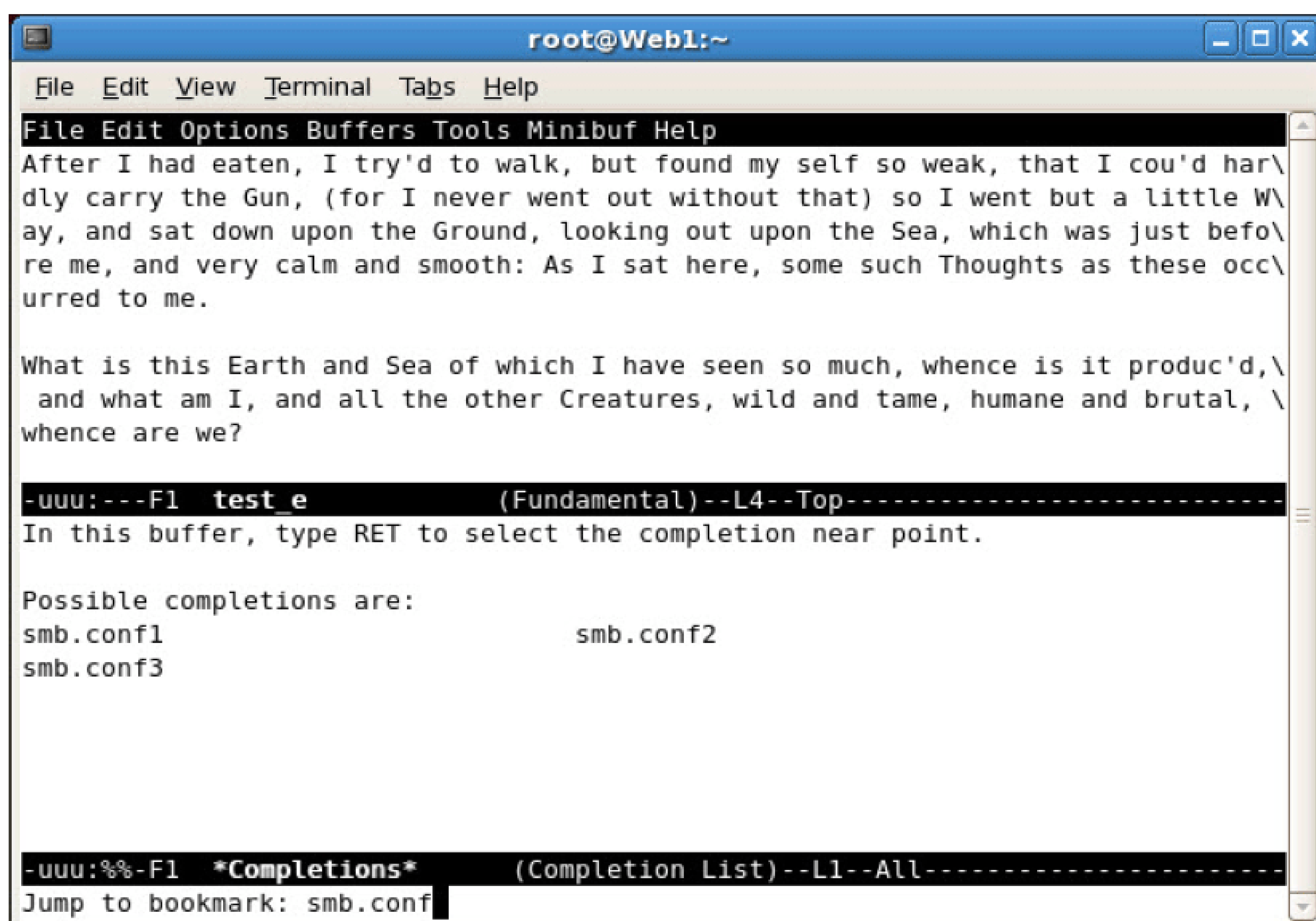
```
File Edit Options Buffers Tools Minibuf Help
After I had eaten, I try'd to walk, but found my self so weak, that I cou'd har\
dly carry the Gun, (for I never went out without that) so I went but a little W\
ay, and sat down upon the Ground, looking out upon the Sea, which was just befo\
re me, and very calm and smooth: As I sat here, some such Thoughts as these occ\
urred to me.


What is this Earth and Sea of which I have seen so much, whence is it produc'd,\
 and what am I, and all the other Creatures, wild and tame, humane and brutal, \
whence are we?

Sure we are all made by some secret Power, who form'd the Earth and Sea, the Ai\
r and Skyd?B àmov'd, all the Impression which was rais'd from it, wore off also\
, as I have noted already.

Even the Earthquake, tho' nothing could be more terrible in its Nature, or more\
 immediately directing to the invisible Power which alone directs such Things, \
yet no sooner was the first Fright over, but the Impression it had made went of\
f also. I had no more Sense of God or his Judgments, much less of the present A\
ffliction of my Circumstances being from his Hand, than if I had been in the mo\
st prosperous Condition of Life.
-uuu:**-F1  test_e          (Fundamental)--L3--Top--------------------------
Jump to bookmark:
```

图 12.16　跳转到书签



```
File Edit Options Buffers Tools Minibuf Help
After I had eaten, I try'd to walk, but found my self so weak, that I cou'd har\
dly carry the Gun, (for I never went out without that) so I went but a little W\
ay, and sat down upon the Ground, looking out upon the Sea, which was just befo\
re me, and very calm and smooth: As I sat here, some such Thoughts as these occ\
urred to me.

What is this Earth and Sea of which I have seen so much, whence is it produc'd,\
 and what am I, and all the other Creatures, wild and tame, humane and brutal, \
whence are we?

-uuu:---F1  test_e          (Fundamental)--L4--Top--------------------------
In this buffer, type RET to select the completion near point.

Possible completions are:
smb.conf1                          smb.conf2
smb.conf3



-uuu:%%-F1  *Completions*     (Completion List)--L1--All----------------------
Jump to bookmark: smb.conf
```

图 12.17　书签名补全功能

在图 12.17 中，Emacs 编辑器在小缓冲区的上方显示了当前所有书签名称，并在小缓冲中补全了书签名中相同的部分。此时用户输入需要的书签名，并按下 Enter 键就可以跳转到相应的书签位置。

3．管理书签

当熟练使用书签之后，可能会保存许多书签，这时需要对书签进行管理。常见的管理内容有：重命名书签、删除书签等。

（1）如果要删除书签，可以使用命令 bookmark-rename。按下 Esc x，然后输入命令

bookmark-rename，此时 Emacs 会要求用户输入要修改的书签名，如图 12.18 所示。

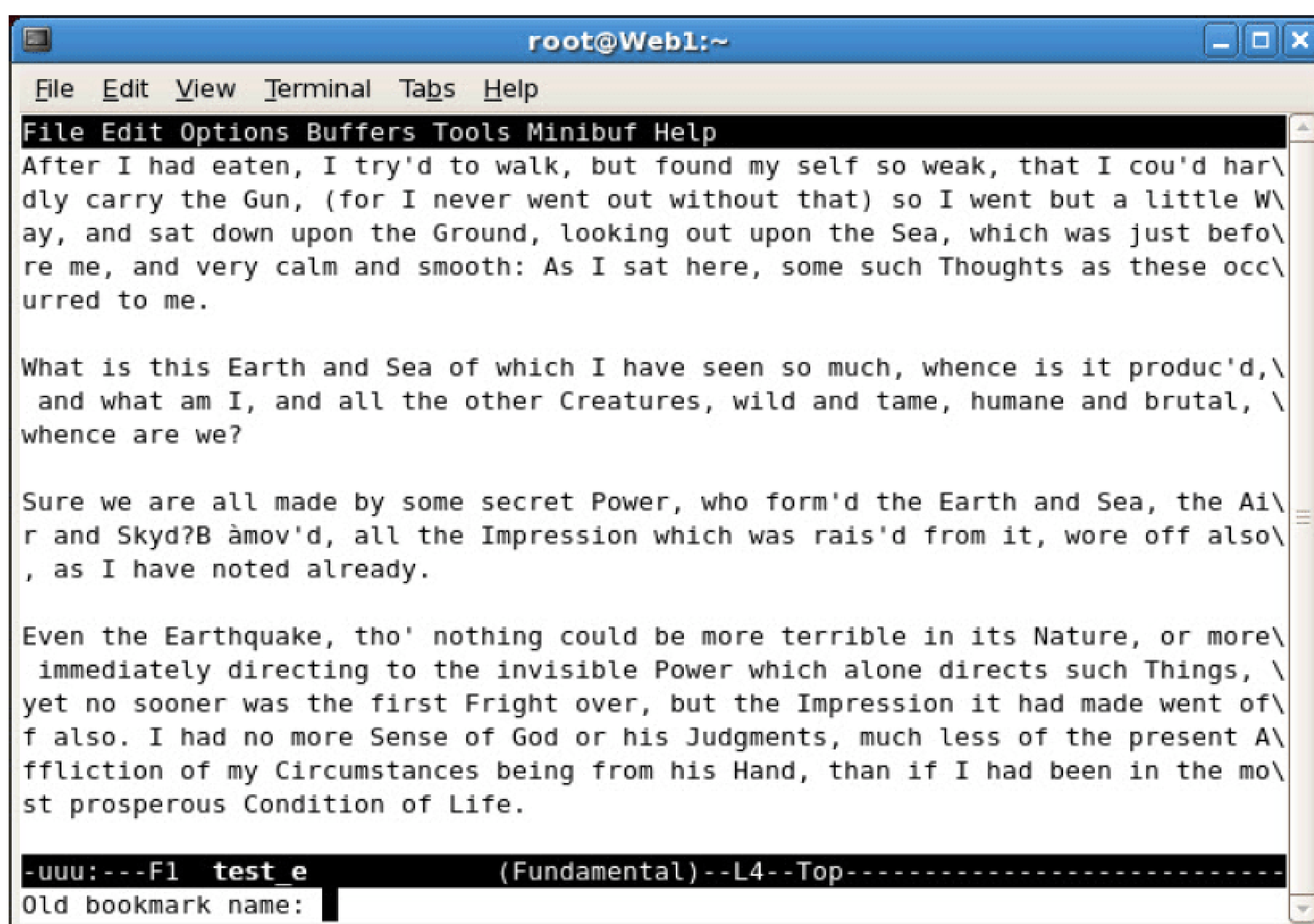


图 12.18　修改书签名

在图 12.18 中，Emacs 将光标移动到小缓冲区内，并要求用户输入旧书签名。此时可以输入需要修改的书签名，或使用书签名补全功能查找需要修改的书签名。完成后按 Enter 键，Emacs 会提示用户输入新名称，在 New name 之后输入新书签名并按 Enter 键就可以完成修改。

（2）当不再需要使用某个书签时，可以使用命令 bookmark-delete。使用命令后输入需要删除的书签名就可以完成删除。

（3）有时删除、重命名了某个书签的目标文件后，访问书签会提示如图 12.19 所示错误。

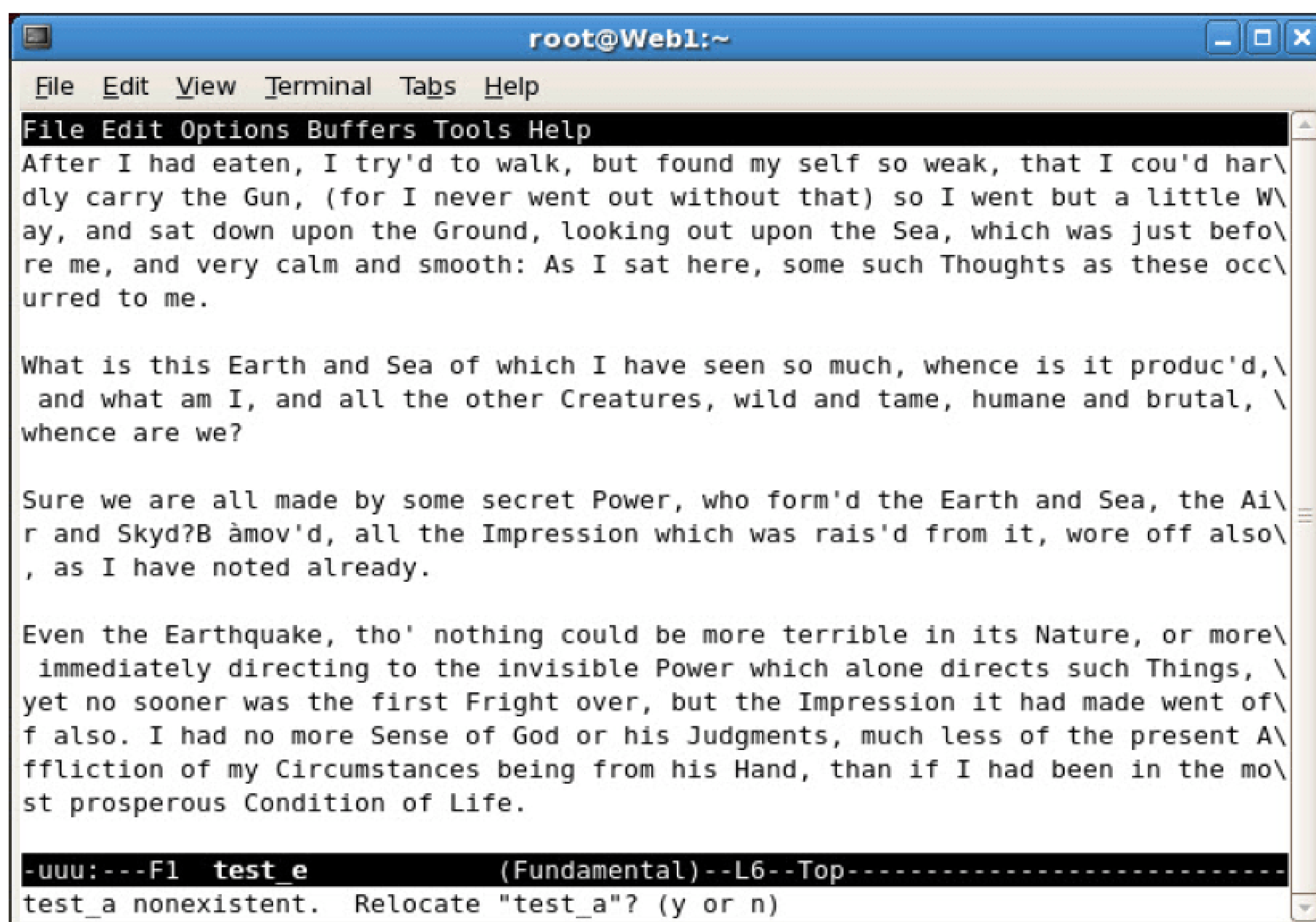


图 12.19　书签错误

在图 12.19 中，Emacs 在小缓冲区中提示目标文件不存在，并询问用户是否需要更新书签。此时可以执行的操作如下。

- ❑ 按 y 键更新文件的新位置。当目标文件被重命名时，可以使用这个操作。
- ❑ 按 n 键不更新。这个操作将不更新书签，访问书签的操作将会中止。

需要注意的是按 n 键选择不更新书签时，Emacs 不会自动删除失效的旧书签，因此需要用户手动删除。

（4）如果需要查看目前保存的所有书签列表，可以使用快捷键 Ctrl+x r l。使用此命令后，Emacs 会列出所有保存的书签列表，如图 12.20 所示。

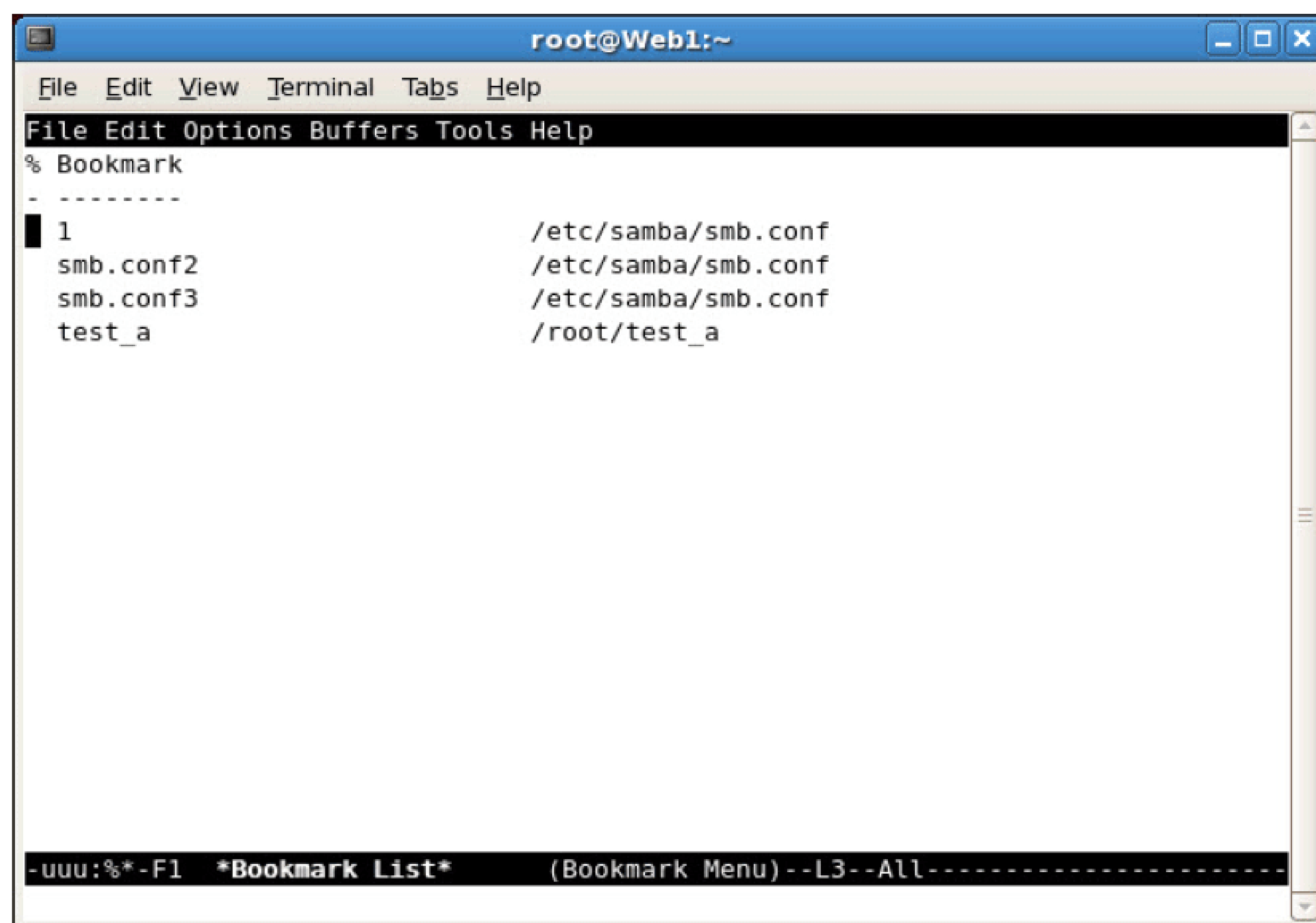


图 12.20　Emacs 的书签列表

在图 12.20 中，Emacs 列出了当前保存的所有书签列表。在这个列表中，可以进行的操作如下。

- ❑ 使用上、下方向键和 d 键给书签加上待删除标记。标记完成后，可以使用 x 键删除书签。
- ❑ 使用上、下方向键选中书签，并按 r 键重命名书签。
- ❑ 使用上、下方向键选中书签，并按 f 键立即跳转到书签。

除此之外，还有许多不常用的操作，例如使用 u 键删除待操作标记等。读者可以阅读相关文档，此处不再一一介绍。

12.5.4　灾难恢复

任何计算机系统都不是安全的，总有一些情况会导致数据丢失。在使用 Emacs 编辑文本时，有许多情况可能会导致 Emacs 进程结束，例如程序崩溃、主机掉电等。这时就可以使用 Emacs 的灾难恢复功能找回数据。在 Emacs 编辑器中，可以利用两种文件恢复数据：备份文件、自动保存文件。

1．利用备份文件恢复数据

在编辑过程中，每次保存文件后，Emacs 编辑器会自动生成备份文件。例如正在编辑的文件名为 test，那么备份文件名就是 test~（有时也会存在许多个 test~*n*~，*n* 为数字）。Emacs 编辑器生成备份文件的目的是为了防止发生编辑操作过多，以至于使用撤销功能都无法撤销的情况。

由于 Emacs 生成的仅仅是一个备份文件，因此可以直接使用 Emacs 打开备份文件：

```
#使用 Emacs 打开备份文件
# emacs test~
```

打开备份文件并检查无误后，可以直接使用 Ctrl+X Ctrl+W 将文件另存。

2．利用自动保存的文件恢复数据

与大多数编辑器一样，Emacs 编辑器也会自动保存当前正在编辑的文件。每过一段时间，Emacs 会自动保存缓冲区中的内容，以防止程序崩溃或主机掉电等情况。如果当前编辑的文件名为 test，那么自动保存的文件名就是#test#。

使用自动保存文件恢复数据之前，应该确保要恢复的文件有自动保存的文件。如果没有，就无法使用自动保存文件恢复数据。要恢复数据，可以使用命令 recover-file，使用此命令后 Emacs 会提示用户输入自动保存文件的文件名。输入自动保存的文件名后，按 Enter 键就可以读取自动保存的文件。

读取到自动保存的文件后，可以先将恢复的数据另存，比较、检查之后再覆盖原文件。

12.5.5　使用多窗口

与 Vim 编辑器一样，Emacs 编辑器也有单窗口的不方便之处。例如单窗口非常不方便比较两个文件的内容，或者是通过比较两个文件的内容进行编辑。本小节将简单介绍如何在 Emacs 编辑器中使用多窗口解决这些问题。

1．创建多个窗口

在 Emacs 编辑器中，创建窗口的方式有多种，其实质都是将当前编辑器的窗口分割为不同的几个窗口。按分割的方式可以分为创建上、下窗口、左、右窗口等。

（1）要创建将当前窗口分割为上、下两个窗口可以使用快捷键 Ctrl+X 2。分割后的 Emacs 如图 12.21 所示。

在图 12.21 中，Emacs 编辑器被分为上、下两个窗口（这两个窗口中的内容完全一样），并且 Emacs 还会自动将光标移动到上面的窗口中。

（2）如果要继续将窗口分割为左、右两个窗口，可以使用快捷键 Ctrl+X 3。分割后的 Emacs 如图 12.22 所示。

在图 12.22 中，Emacs 编辑器上面的窗口被分割为左、右两个窗口，这两个窗口之间使用竖线作为分隔符。

使用 Emacs 编辑器时，也可以使用这种继续分割的方法将 Emacs 分割为多个不同的窗口。

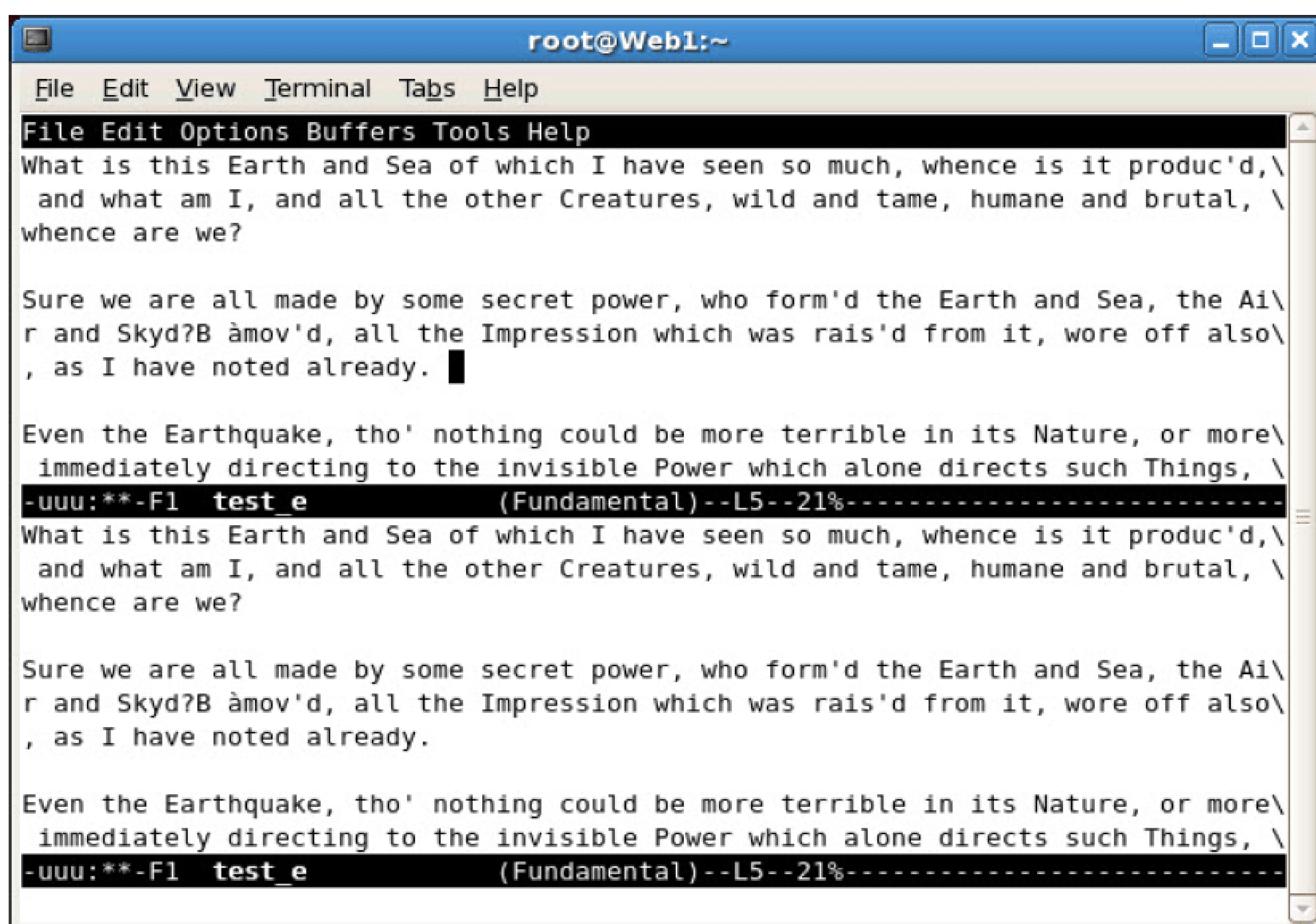


图 12.21　将 Emacs 分割为上、下两个窗口

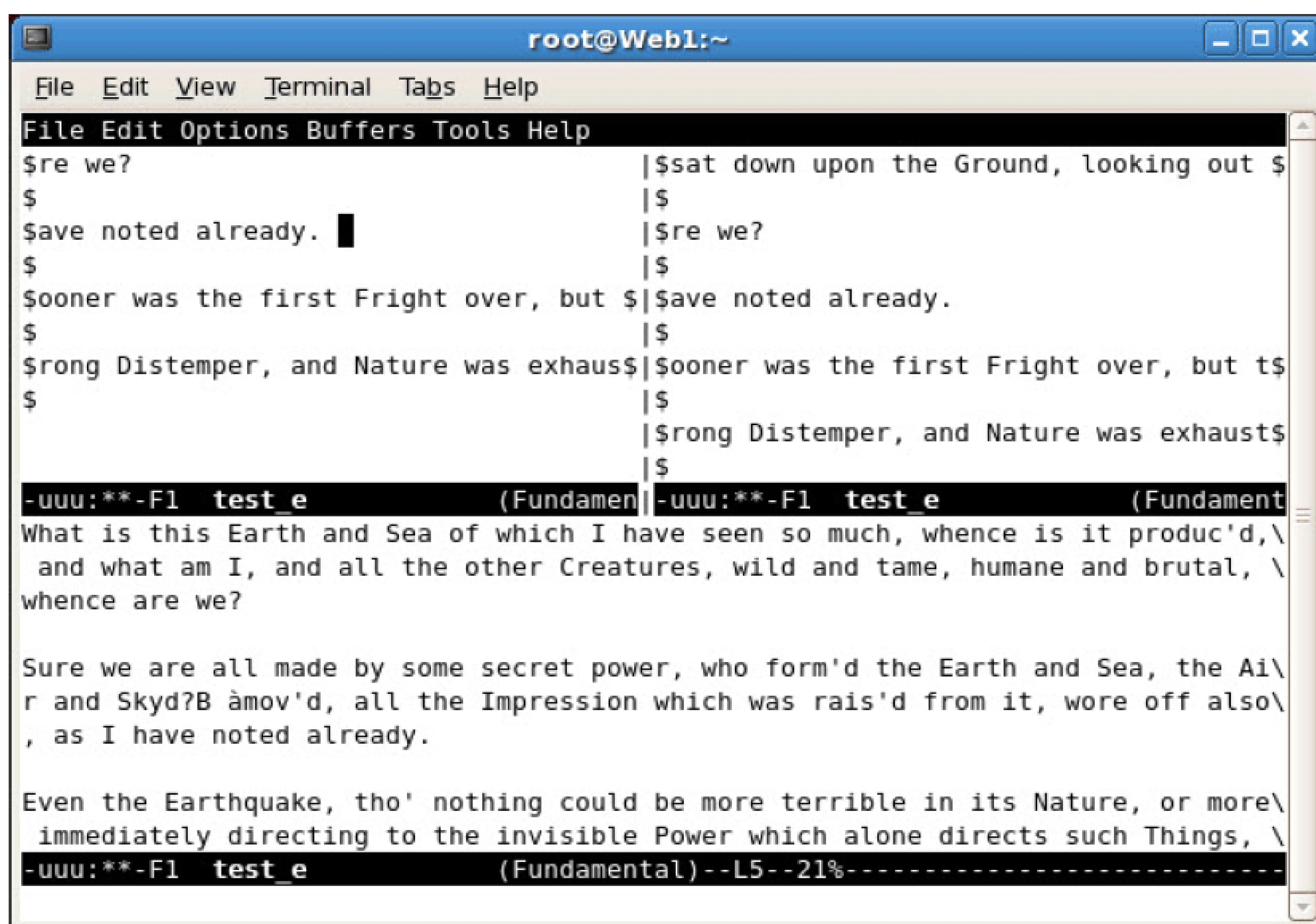


图 12.22　继续分割窗口

（3）也可以在 Emacs 编辑器启动时打开多个窗口，操作方法是将多个文件作为 emacs 命令的参数：

```
#使用 Emacs 编辑器同时编辑 a1、a2 两个文件
# emacs -nw a1 a2
```

使用上面这个命令时，Emacs 将在启动后自动打开两个窗口，并在这两个窗口中编辑不同的文件。

提示：使用分割的方法打开多个窗口时，可以使用打开新文件的方法（例如使用 Ctrl+X Ctrl+F 打开文件）在窗口中编辑另一个文件。

2．切换窗口

打开了多个窗口之后，为了在不同的窗口中执行编辑，需要切换光标至不同的窗口。通常可以使用快捷键 Ctrl+X O 将光标切换到其他窗口，切换后就可以对相应的窗口进行编辑了。

3．关闭窗口

当不再需要某个窗口时，要关闭这些窗口。关闭窗口的快捷键如下。

- ❑ Ctrl+X 0：关闭当前光标所在窗口。
- ❑ Ctrl+X 1：关闭除当前窗口之外的所有窗口。

使用这两个快捷键关闭窗口之前，一定要检查窗口中的文本是否都已经保存。这是因为窗口虽然关闭，但窗口使用的缓冲区还会存在，这种情况有可能会导致数据丢失。

4．窗口与缓冲区

使用 Emacs 编辑文本时，Emacs 总是将编辑的内容保存在缓冲区中，所有的编辑动作都在缓冲区中进行，直到使用保存命令为止。这在多窗口中有所不同，这是因为 Emacs 编辑器可能会在不同的窗口中使用了不同的缓冲区。例如图 12.23 中所示的 Emacs 就使用了多个缓冲区。

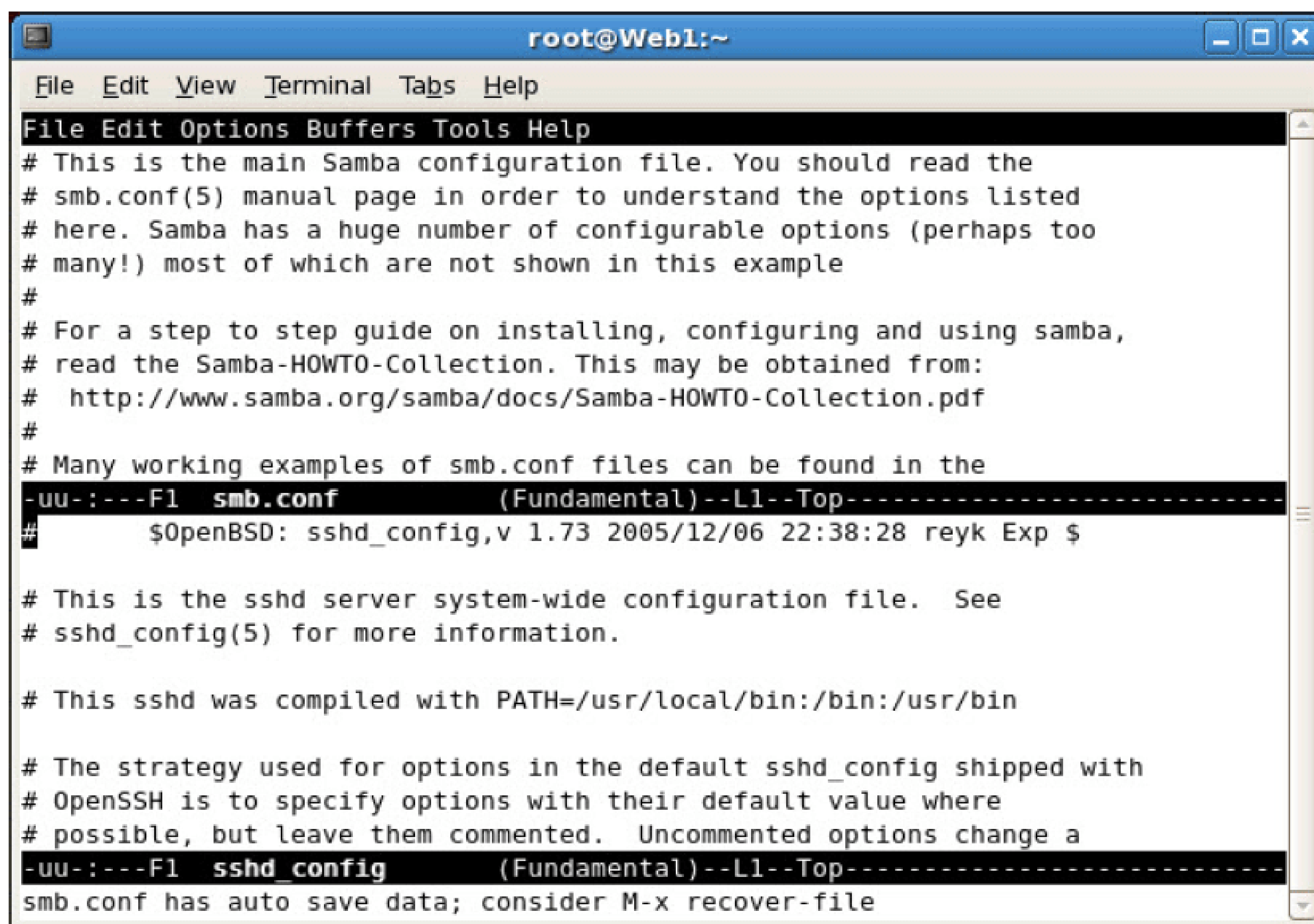


图 12.23　多窗口中使用不同的缓冲区

在图 12.23 中，Emacs 编辑器打开了两个窗口，并且这两个窗口使用了不同的缓冲区（状态栏中的文件名就是缓冲区名称）。这可能会带来一些麻烦，例如当我们关闭其中一个窗口时，Emacs 并不会提示使用者保存缓冲区中的内容。

像上面这种情况就可以使用缓冲区列表来解决。使用快捷键 Ctrl+X Ctrl+B 打开缓冲区列表，如图 12.24 所示。

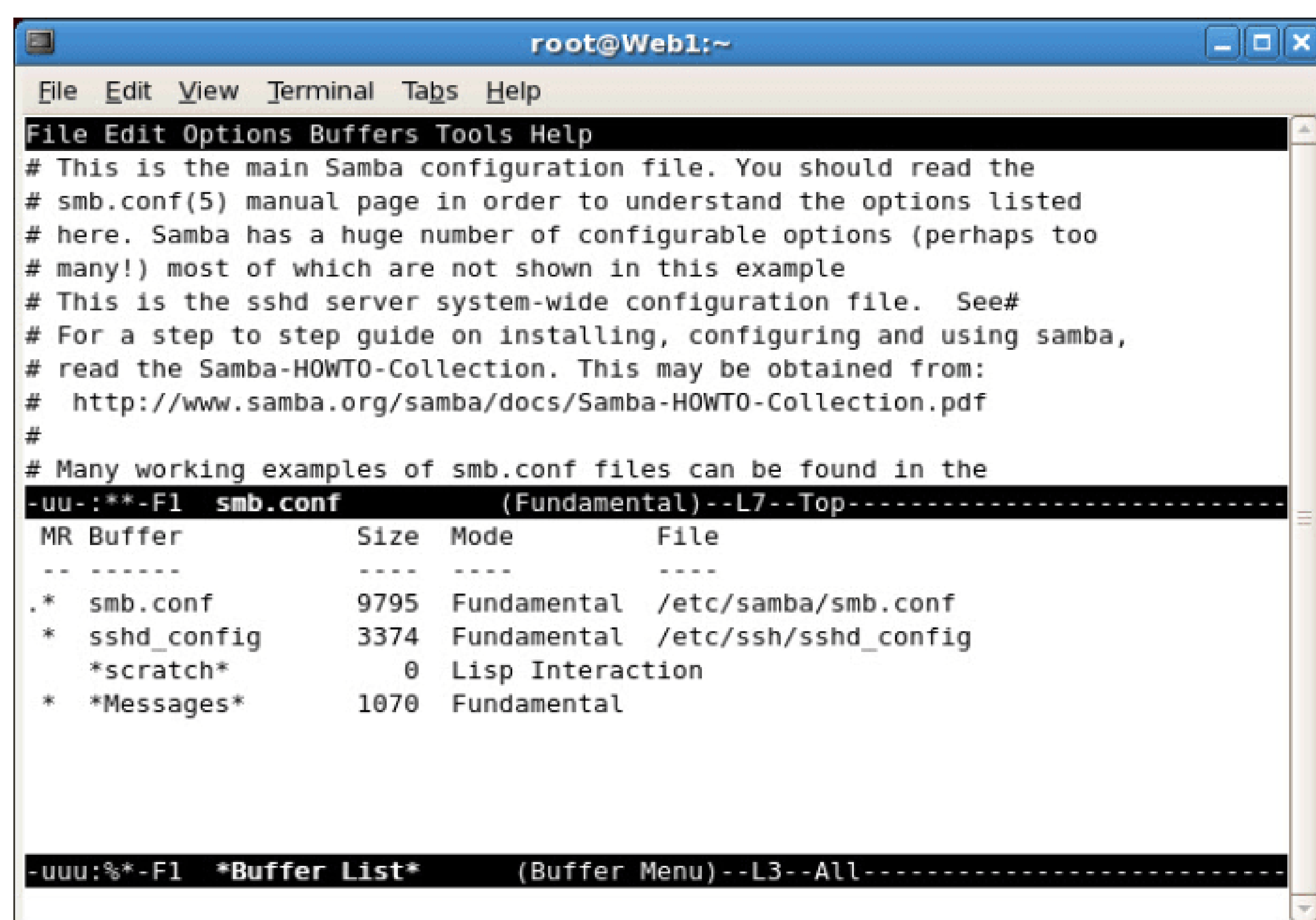


图 12.24　缓冲区列表

在图 12.24 所示的窗口中，有一个名为 Buffer List 的窗口，这个窗口就是当前使用的缓冲区列表。缓冲区列表以一个表的形式显示，分为 MR（标记）、Buffer（缓冲区名称）、Size（大小）、Mode（模式）、File（缓冲区中保存的文件）。

缓冲区列表中的 MR 字段是一个标记字段，常见的标记及含义如下。

- .：当前正在显示。
- *：这个缓冲区中的文本已经修改。
- %：缓冲区中的文本为只读。

上面的标记中要特别注意的是已经修改、但又未显示的缓冲区，这些缓冲区可能需要保存。

【操作缓冲区列表】

使用 Ctrl+X O 将光标移动到缓冲区列表窗口中，然后使用以下方式操作缓冲区列表。

- 使用方向键和 k、d 键为缓冲区加上待删除标记（待删除标记将在 MB 字段中显示，标记为 D），然后使用 x 键删除缓冲区。
- 使用方向键和 s 键加上待保存标记（在 MB 字段中用 S 表示），然后使用 x 键保存。
- 使用方向键和 1 键（数字 1），全屏显示当前缓冲区。这个操作适用于需要编辑某个缓冲区时。
- 使用方向键和 m 键为缓冲区加上待显示标记（在 MB 字段中用>表示），然后使用 v 键显示这些有 m 标记的缓冲区（有多个待显示标记时，Emacs 会在多窗口中显示）。

当然如果加上待操作标记又后悔了，也可以像书签列表、文件列表那样使用 u 键撤销标记。利用这些操作，用户可以对某个缓冲区执行更进一步的操作。

12.6　目 录 模 式

在 Emacs 编辑器中，有一个比较特殊的目录模式。在此模式中，用户可以执行文件操

作，例如重命名文件、复制文件、删除文件等。要进入 Emacs 目录模式，可以使用以下方式。

- 启动文件时，传递给 Emacs 编辑器的是一个目录，而不是一个文件。
- 在 Emacs 中，使用任何打开文件的命令（例如 Ctrl+X Ctrl+F），并指定要打开的是一个目录。
- 使用快捷键 Ctrl+X D，并输入一个目录。

进入目录模式后，Emacs 将显示目录中的文件，如图 12.25 所示。

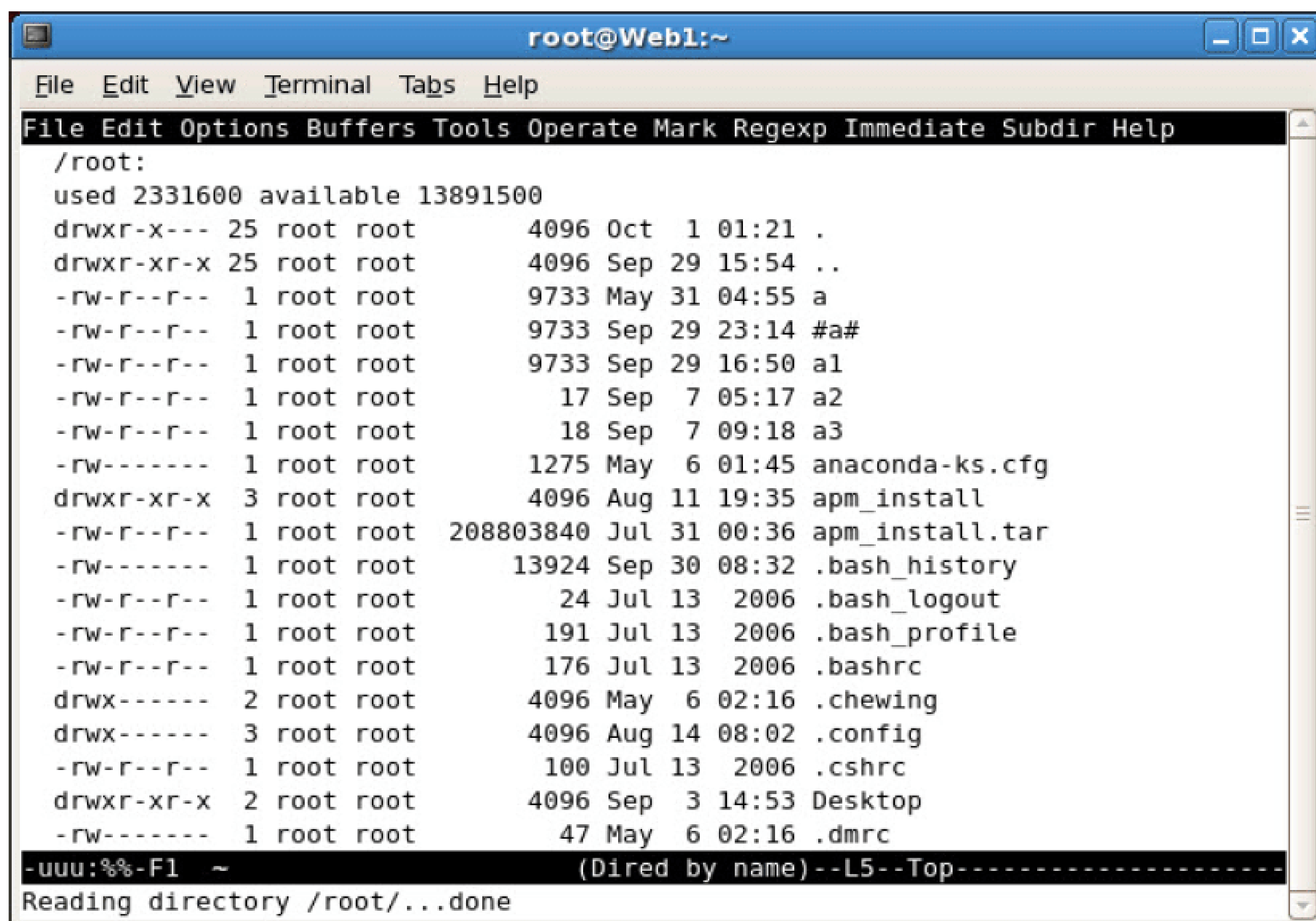


图 12.25　Emacs 的目录模式

从图 12.25 中可以看出，目录模式中的内容与命令 ls-l 的输出非常相似。本节就从这个类似 ls 命令的输出开始介绍 Emacs 编辑器的目录模式。

12.6.1　查看文件

目录模式的首要功能是可以快速浏览、查看文件。本小节将简单介绍如何在目录模式中查看文件。

（1）排序文件

在图 12.25 所示的目录模式中，Emacs 列出了指定目录的所有文件。仔细观察这个文件列表，可以发现这个文件列表是按文件名排序的。如果要按时间排序（这样能更快地找到最近编辑的文件），可以按 s 键。

（2）快速浏览文件内容

有时可能会忘记需要编辑的文件名，这时可以通过快速浏览文件内容的方式查找文件。在目录模式中，使用方向键将光标移动到文件上，然后按下 v 键。此时 Emacs 会显示文件内容，但此时的文件内容是无法编辑的。如果发现当前浏览的文件不是需要编辑的文件，可以按下 Ctrl+C 键或 Q 键返回文件列表继续浏览文件。

（3）快速选择要编辑的文件

如果确定某个文件就是需要编辑的文件，可以将光标移动到文件上，然后按下 f 键。

此时 Emacs 会立即将文件读入缓冲区，用户就可以对文件进行编辑了。

12.6.2　删除、复制、重命名文件

在 Emacs 编辑器的目录模式中，除了快速查看、浏览文件内容之外，还可以对文件进行管理操作。本小节将简单介绍如何在目录模式中操作文件。

1．删除文件

在目录模式的文件列表中，可以使用方向键和 d 键为需要删除的文件加上待删除标记。加上待删除标记后，Emacs 会在行首显示 D，如图 12.26 所示。

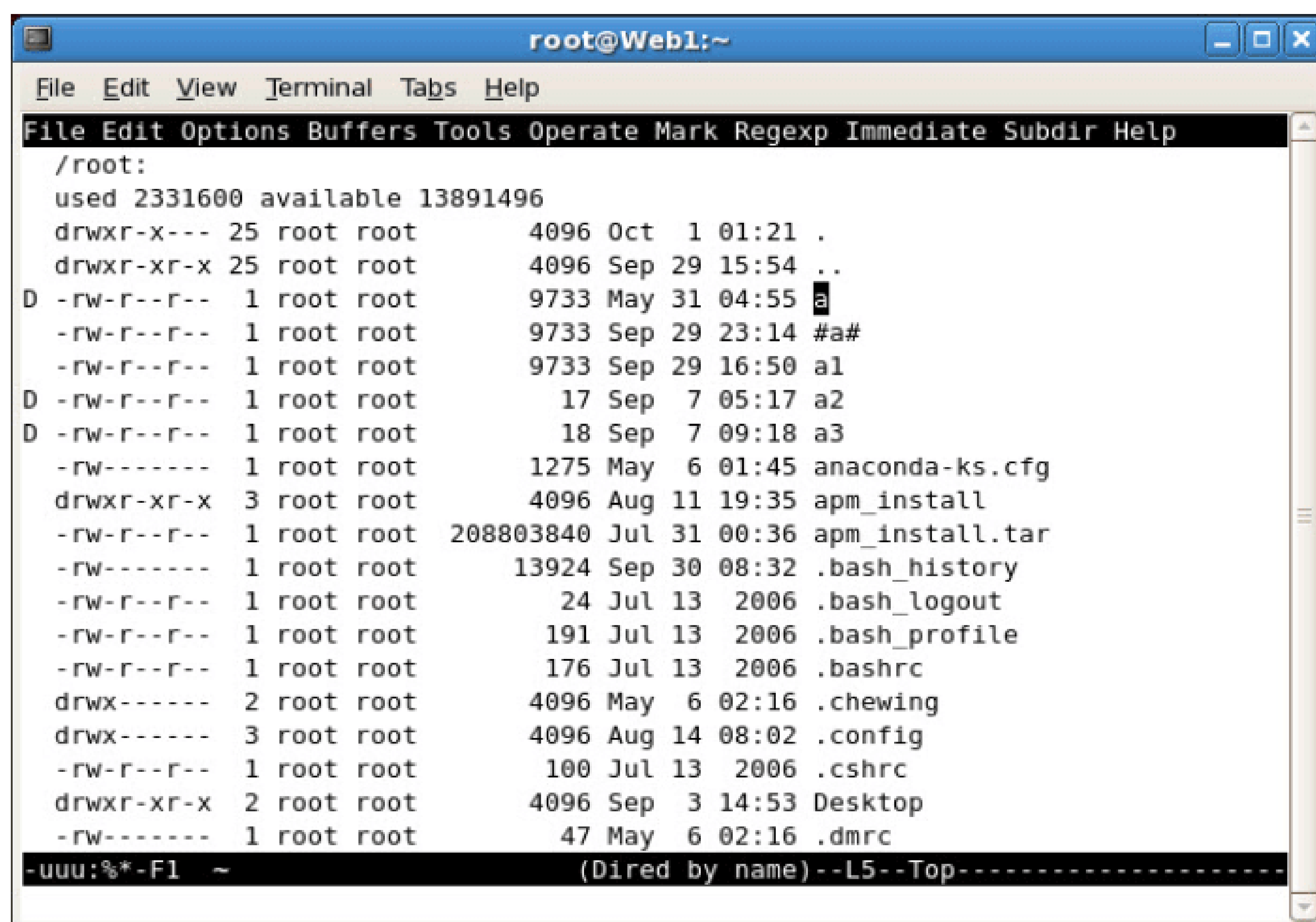


图 12.26　文件的待删除标记

在图 12.26 中，有 3 个文件添加了待删除标记：a、a2 和 a3。如果需要去掉某个文件的待删除标记，可以将光标移动到文件上，然后按下 u 键。

当用户确认要删除所有标记的文件后，就可以按下 x 键，并在稍后的询问中输入 yes 将所有标记的文件删除。

技巧： 如果需要选择所有备份文件和自动保存的文件，可以按下#键和~键为这些文件添加待删除标记。这在清理无用的备份文件和自动保存文件时非常有用。

2．复制文件

在目录模式中复制文件时，可以先将光标移动到需要复制的文件上，然后按下 C 键（必须是大写字母）。此时 Emacs 会提示用户输入目标位置和名称，按需要输入路径和文件名就可以完成复制。

3．重命名文件

重命名文件与书签功能中的重命名类似，先将光标移动到要重命名的文件上，然后按下 R 键（必须是大写字母）。最后按提示输入文件的新名称即可完成重命名。

12.6.3　操作压缩文件

在 Emacs 编辑器的目录模式中，除了可以进行简单的文件操作外，还可以操作压缩文件。本小节将简单介绍如何操作压缩文件。

（1）压缩文件

将光标移动到需要压缩的文件上，然后按下 Z 键（必须是大写字母）。此时 Emacs 编辑器会提示用户是否需要压缩文件，输入 y 并按 Enter 键即可完成文件的压缩任务。

注意：使用目录模式对文件进行压缩时，Emacs 会使用 Gzip 压缩并删除源文件。压缩之后，Emacs 编辑器将无法直接编辑压缩的文件。

（2）解压缩文件

压缩文件时，如果光标指向的文件是一个已经压缩的文件，Emacs 会将压缩文件解压缩，并删除源文件。

注意：Emacs 编辑器以文件名识别压缩文件，因此无法解压修改了文件名的压缩文件。Emacs 编辑器可以识别的压缩格式有.gz、.Z、.tgz 等。

12.6.4　其他文件操作

在 Emacs 编辑器的目录模式中，除了前面几个小节中介绍的操作外，还可以执行其他操作，本小节将简单介绍这些操作。

1．比较文件内容

在目录模式中，可以使用一个比较简单的方式实现比较文件内容操作，比较文件内容功能与 diff 命令相同。

先将光标移动到需要比较的文件上，然后按下=键。此时 Emacs 会提示用户输入需要比较的另一个文件的名称，输入之后就可以开始执行比较，并将比较结果显示出来，如图 12.27 所示。

在图 12.27 中，显示了两个简单文件 a2 和 a3 的比较结果。

2．将文件作为Shell命令的参数

在目录模式中还可以将文件作为某个 Shell 命令的参数。下面将以简单筛选为例介绍其用法。

（1）将光标移动到需要执行命令的文件 a1 上，然后按下!键。此时 Emacs 会在小缓冲区中提示用户输入要执行的命令。

（2）在本例中需要在 a1 中查找并显示所有含有字符串 home 的行，此时可以输入命令 grep “home”，并按回车键。Emacs 会显示命令的执行结果，如图 12.28 所示。

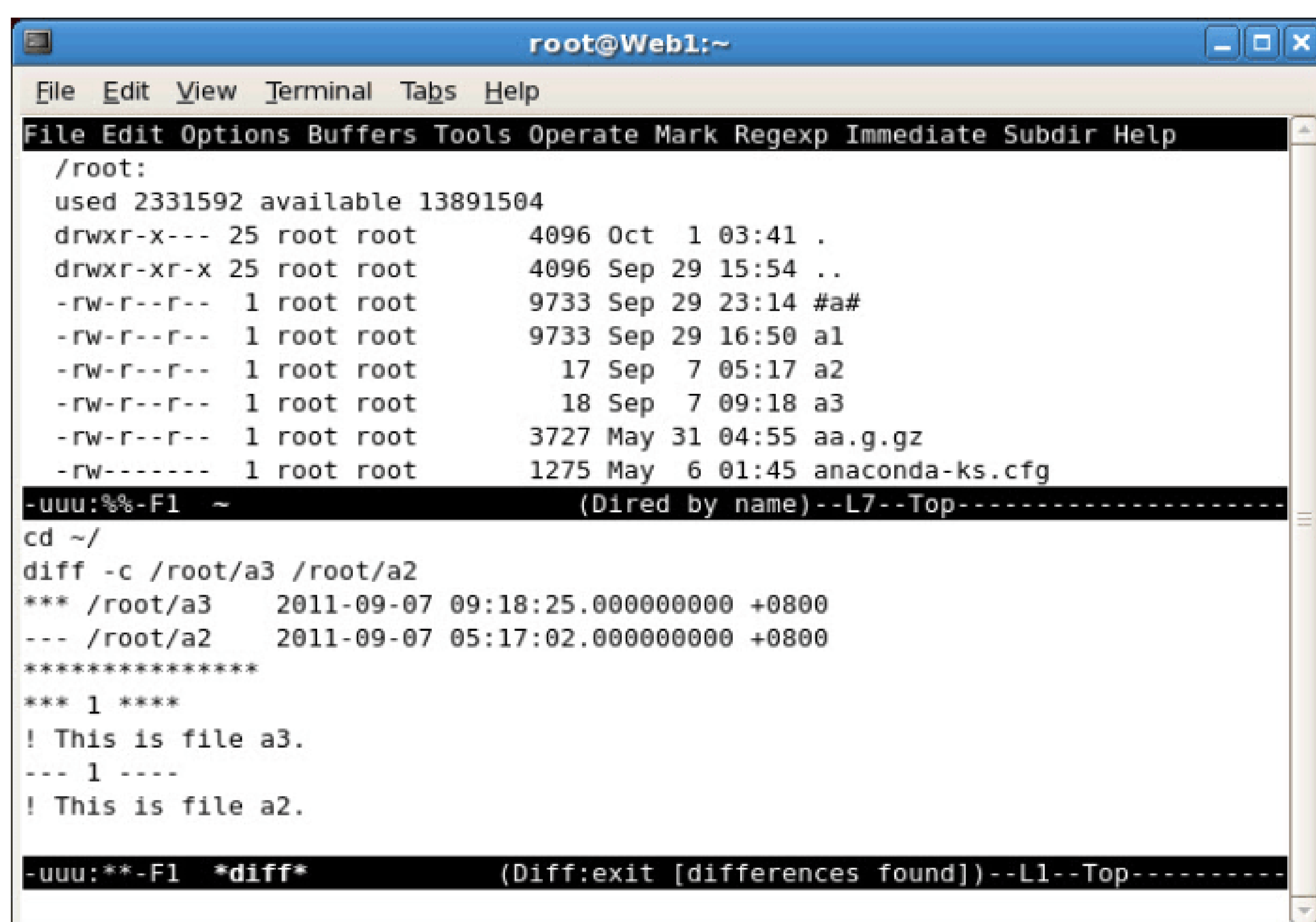


图 12.27　比较文件内容

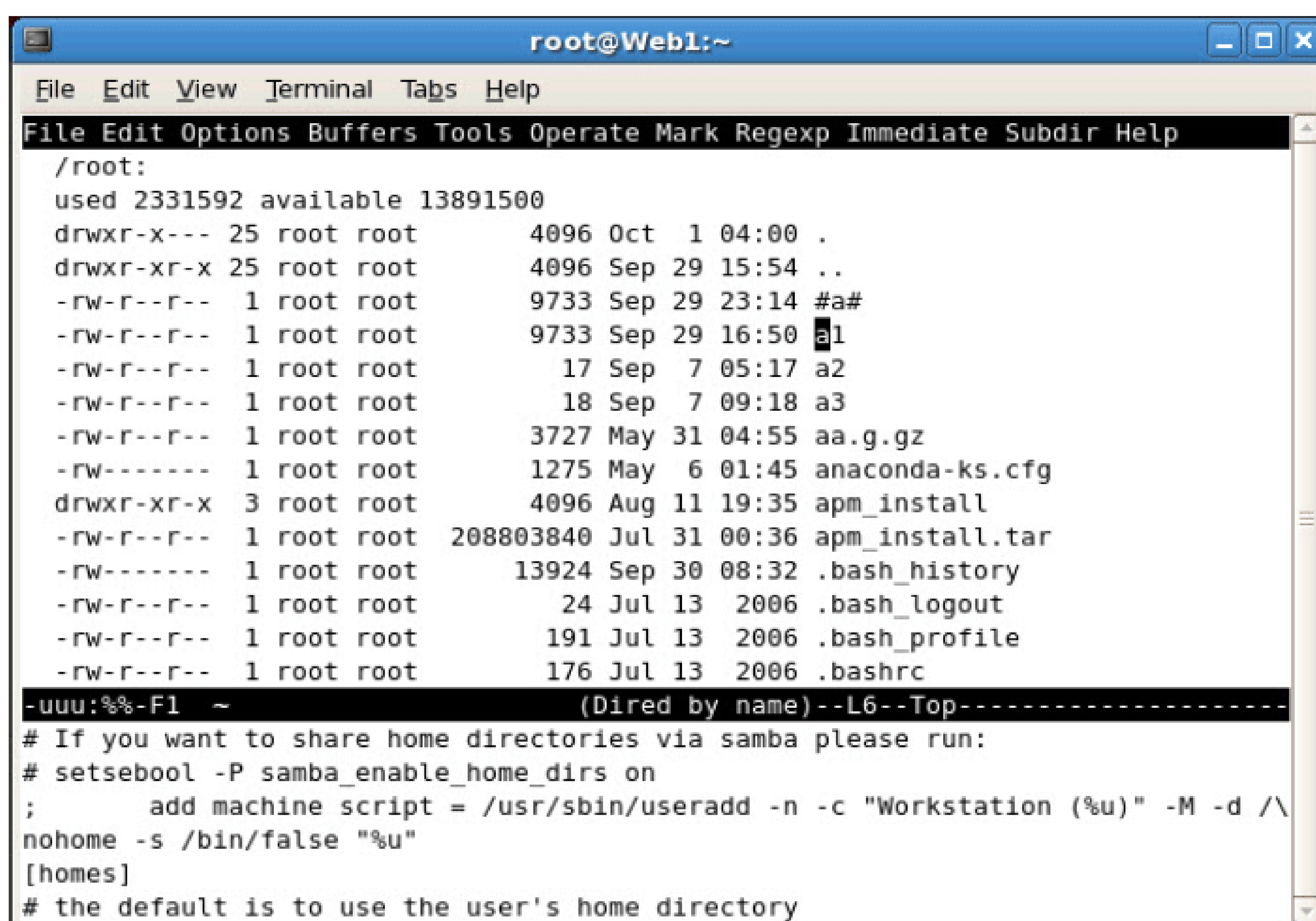


图 12.28　在目录模式中执行命令

从图 12.28 中可以看到，Emacs 执行命令 grep 后，将命令的结果显示在小缓冲区中（关于命令输出的内容，请参考本书 12.7.1 节中的相关内容），此时只需要移动光标就可以去掉显示的结果。

注意：本节仅简单介绍了目录模式中一些较为常见的操作，感兴趣的读者可以阅读相关文档了解更多内容。

12.7　Emacs 编辑器的其他功能

在前面几节中，简单介绍了 Emacs 编辑文本时的常用功能。这些仅仅是 Emacs 编辑器的小部分功能，除了这些之外，Emacs 还可以收发电子邮件、玩游戏等。本节将简单介绍 Emacs 编辑文本之外的功能。

12.7.1　在 Emacs 编辑器中执行 Shell 命令

使用 Emacs 编辑器编写脚本时，经常希望能够验证某个命令的功能，这时可以通过在 Emacs 编辑器中执行命令的方法来验证。本小节将简单介绍如何在 Emacs 编辑器中执行 Shell 命令。

1．执行一条Shell命令

如果要验证一个 Shell 命令，可以按下 Esc !。按下此键后，Emacs 编辑器将在小缓冲区内提示用户输入要执行的命令。输入需要执行的命令后，Emacs 会在新窗口中显示命令执行的结果，如图 12.29 所示。

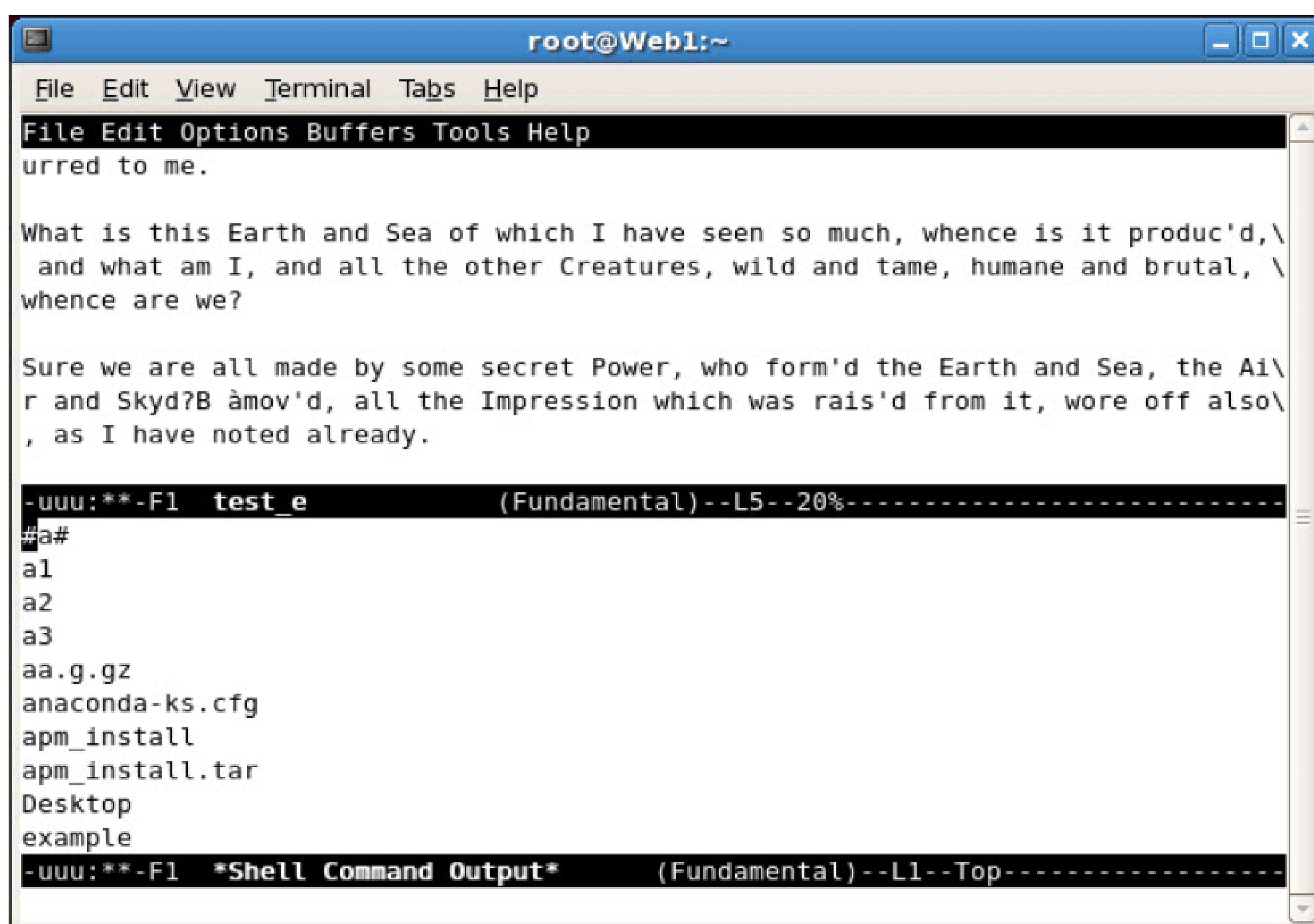


图 12.29　在窗口中显示命令执行结果

在图 12.29 中，Emacs 编辑器新建了一个名为 Shell Command Output 的窗口，并在这个窗口中显示命令的输出。需要注意的是 Emacs 编辑器并不会为每个命令都创建一个窗口，当某些命令的输出较短时，Emacs 会将其输出直接显示在小缓冲区内。只有当命令输出超过一定长度时（通常是 6 行），才会显示在新窗口中。

当命令的输出结果显示在新窗口中时，如果要关闭命令输出显示窗口，可以使用 Ctrl+X1 键（数字 1）。如果要将光标移动到新窗口中，可以使用快捷键 Ctrl+X 0。

2．将命令输出显示在缓冲区中

有时可能希望引用某个命令的输出，执行命令时就应该使用 Ctrl+U Esc !。然后 Emacs 仍然会在小缓冲区内提示用户输入命令。不同的是命令的执行结果将会插入缓冲区中的光标处。

12.7.2　发送电子邮件

Emacs 编辑器除了能编辑文本外，还有许多功能，例如发送、阅读电子邮件，阅读新闻等。本小节将简单介绍 Emacs 的发送电子邮件功能。

要进入编辑邮件模式，可以使用快捷键 Ctrl+X M。使用此快捷键后，Emacs 编辑器将新建一个名为 mail 的窗口，如图 12.30 所示。

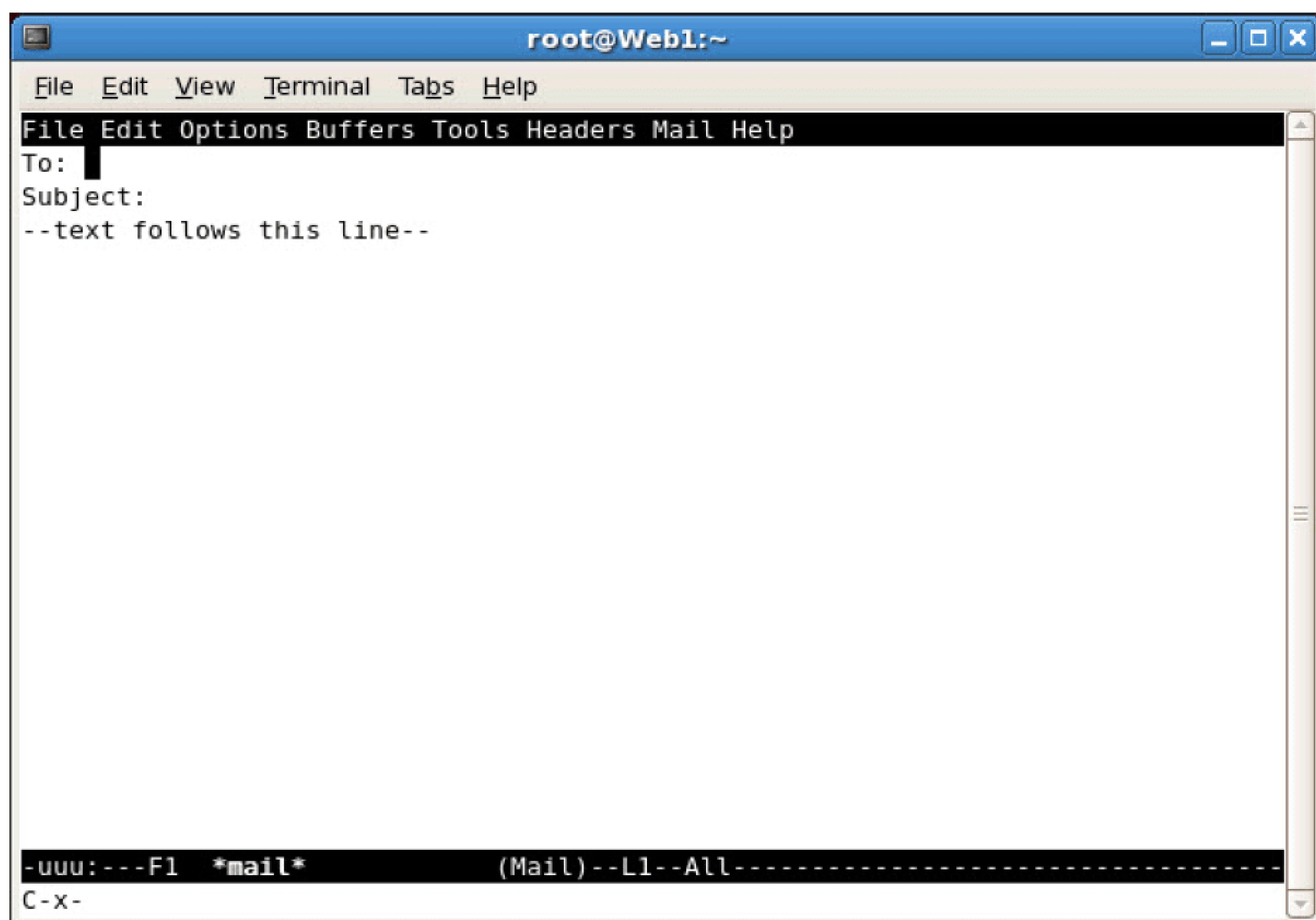


图 12.30　使用 Emacs 发送电子邮件

在图 12.30 所示窗口中，用户可以在 To:后面输入收件人的电子邮件地址。如果有多个地址，可以使用逗号分隔。如果收件人地址多到一行都写不下，可以在第二行继续写，但要在开头加上一个空格。输入收件人的电子邮件地址后，还需要在 Subject:后面输入邮件的主题。

完成上述步骤之后，用户需要在“--text follows this line--”这一行后面（注意不要破坏此行中的内容，否则 Emacs 将无法识别），另起一行开始输入邮件的内容。邮件的内容输入完成后，按下快捷键 Ctrl+C Ctrl +C 就可以将邮件发送出去。

技巧： 用户可以在填写好收件人地址后，另起一行使用 CC:（抄送）和 BCC:（密送）开头添加更多收件人。

12.7.3　阅读电子邮件

既然 Emacs 编辑器可发送电子邮件，就一定可以阅读电子邮件。本小节将简单介绍如何使用 Emacs 编辑器阅读电子邮件。

1．阅读电子邮件

要阅读电子邮件，可以使用命令 rmail。使用该命令后，Emacs 会首先读取用户家目录中的邮件文件 mbox，然后读取/var/spool/mail 中的新邮件。Emacs 编辑器的阅读邮件窗口如图 12.31 所示。

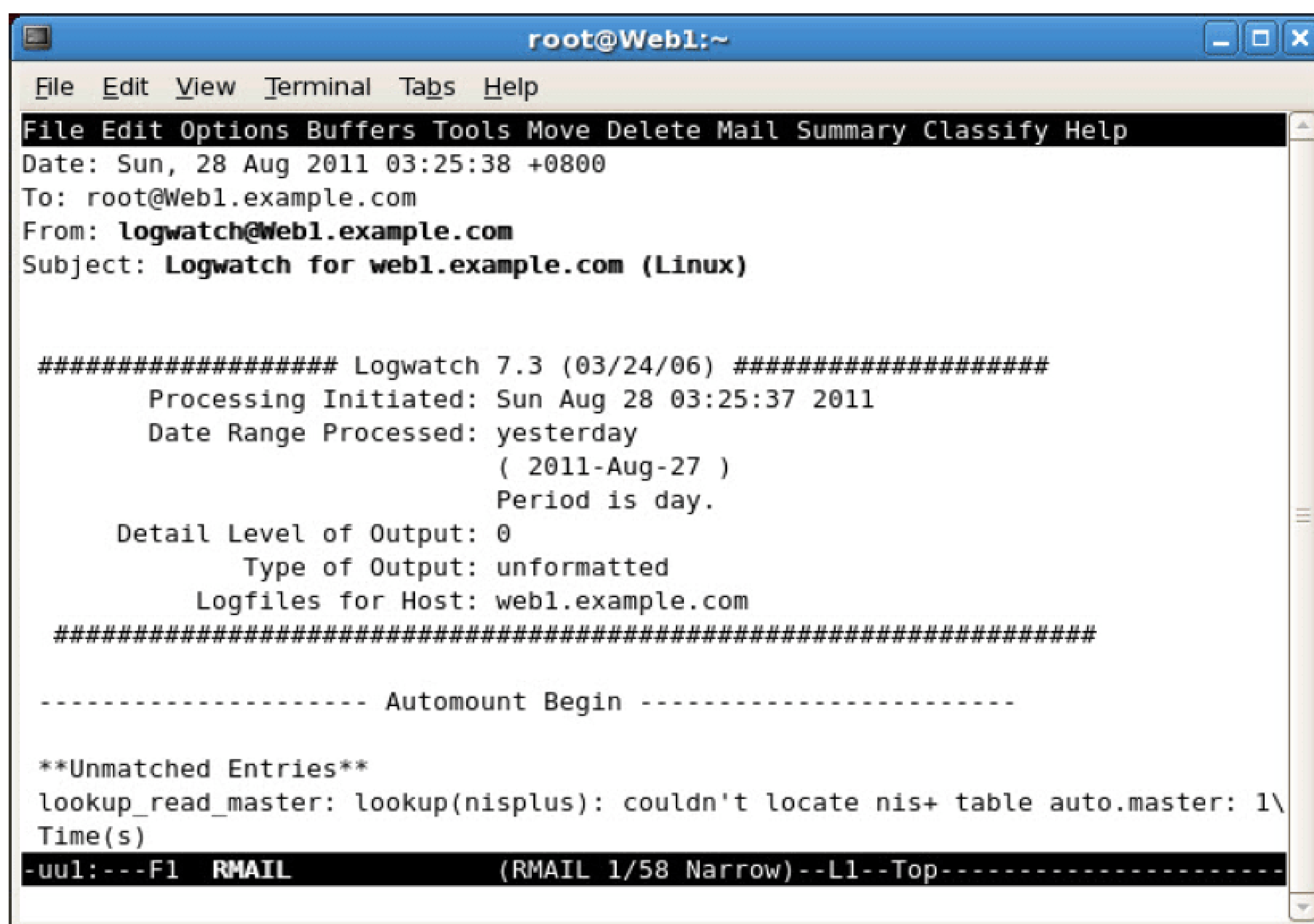


图 12.31　使用 Emacs 编辑器阅读电子邮件

在图 12.31 所示的电子邮件界面中，小缓冲区上面的“1/58”表示当前有 58 封电子邮件，正在阅读第 1 封。

在电子邮件界面中，可以使用的操作如下。

- 空格：滚动到下一屏幕。
- Page Up、Page Down：滚动到上一屏、下一屏。
- .（点号）：移动到邮件头。
- <、>：查看第一条、最后一条邮件。
- n、p：查看下一条、上一条邮件。

由于 Emacs 编辑器读取邮件时先读取用户家目录中的 mbox，因此新邮件通常位于最后。

2．电子邮件列表

如果觉得前面查看邮件的方法不方便，可以按下 h 键打开 RMAIL-summary 窗口查看邮件列表，如图 12.32 所示。

在图 12.32 中，Emacs 被分割为两个窗口，上面的窗口用于显示邮件列表，下面的窗口用于显示邮件的内容。用户可以使用快捷键 Ctrl+X O 在这两个窗口间切换光标，并查看列表和邮件内容。

技巧：可以像前面的书签列表、文件列表那样，在邮件列表中使用 d、u、x 键添加、去掉待删除标记并删除邮件。

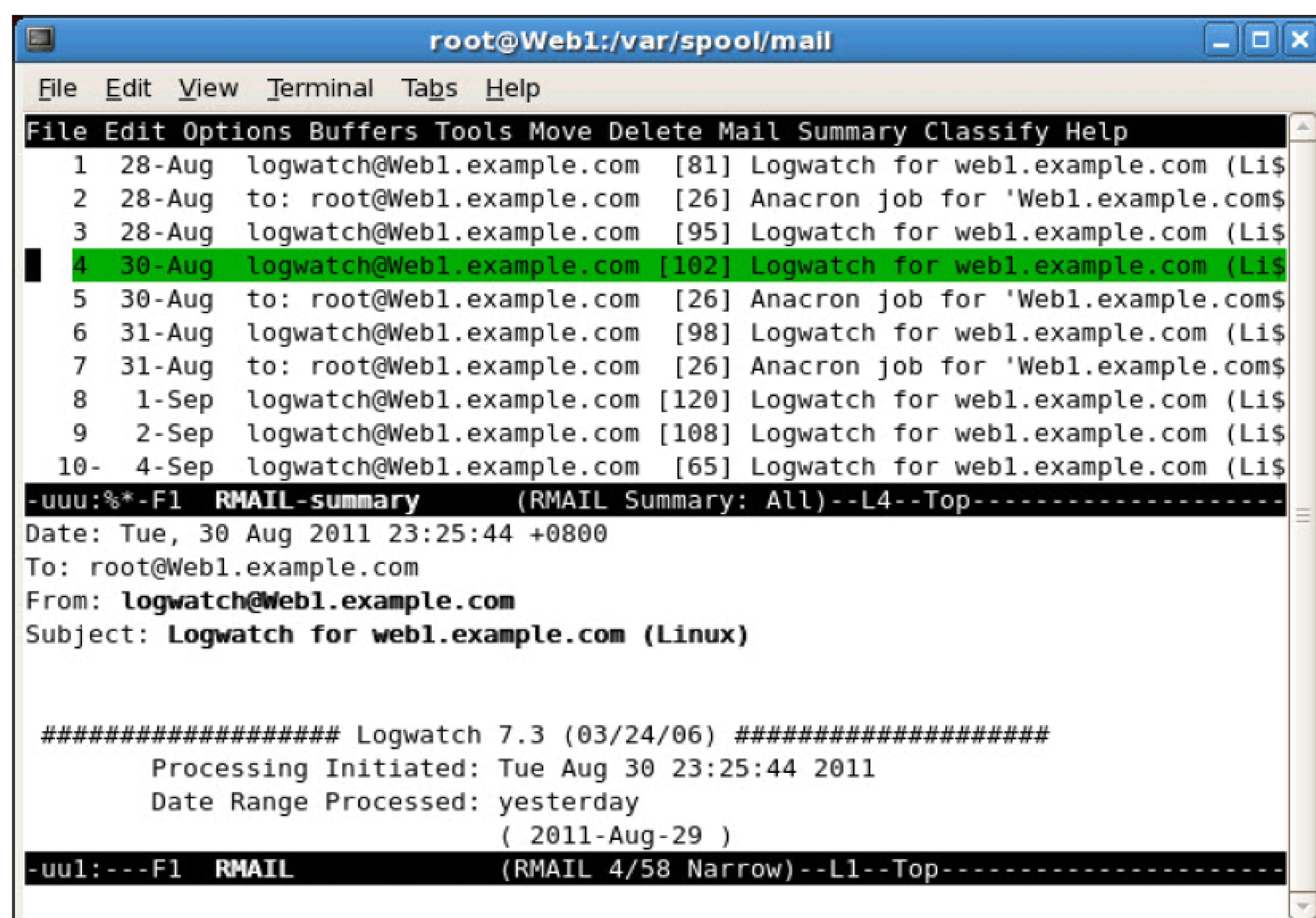


图 12.32　邮件列表

3．从当前邮件中回复

查看邮件时，如果需要回复当前邮件，可以按下 r 键，如图 12.33 所示。

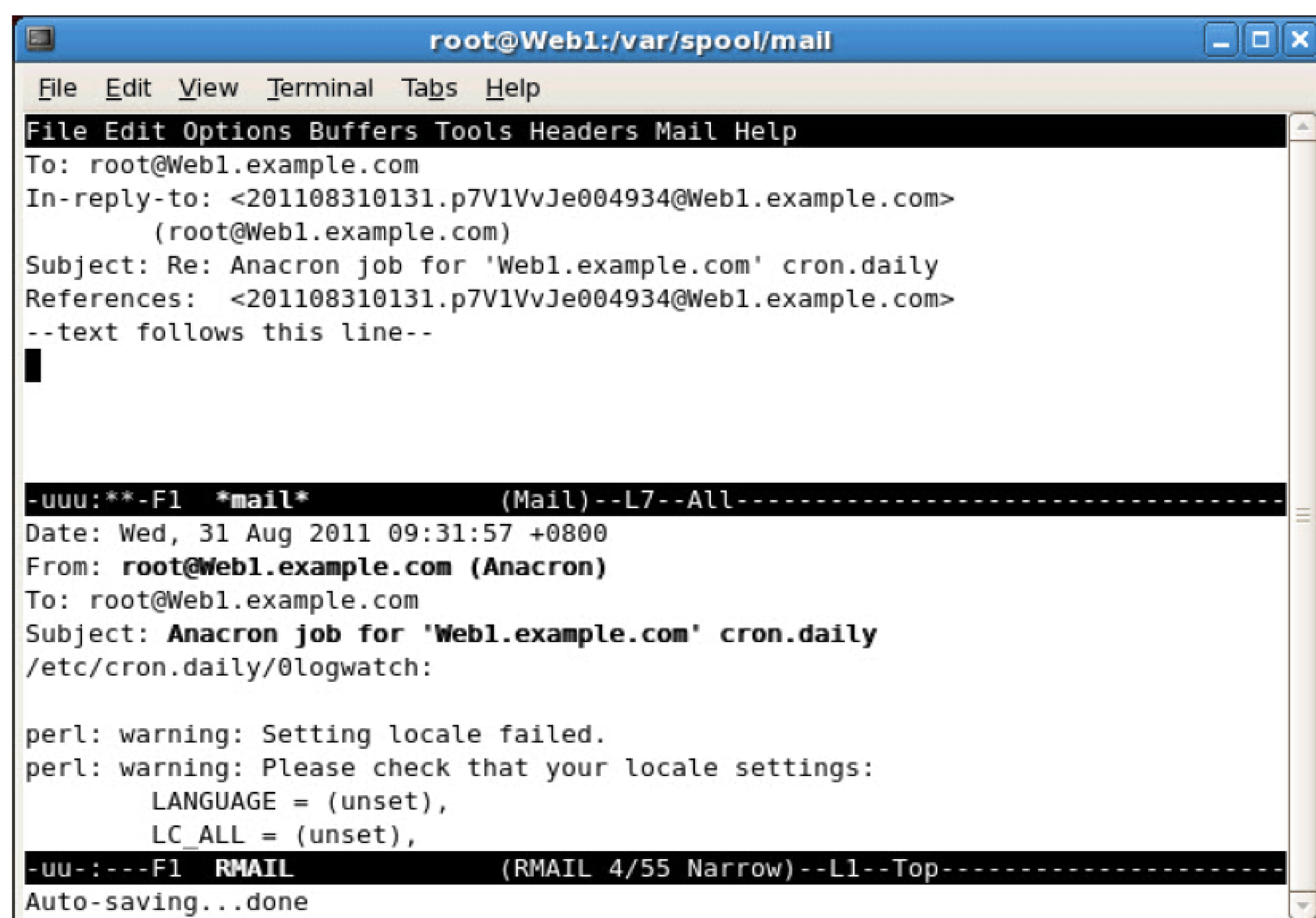


图 12.33　回复邮件

在图 12.33 所示的回复邮件窗口中，Emacs 会自动填写邮件的收件、地址、主题等内容，用户可以直接在“--text follows this line--”这一行后面添加邮件正文。如果要在回复的邮件中插入邮件内容，可以按下 Ctrl+C Ctrl+Y 键，被回复的邮件的正文就会被插入到当前光标处。

12.8 小　　结

- ❑ 12.1 节简单介绍了 Emacs 编辑器的起源、功能概述、如何启动和退出 Emacs，以及 Emacs 界面介绍等内容。
- ❑ 12.2 节主要介绍了 Emacs 的基本使用方法，如何使用菜单栏、打开文件、保存文件等最基本的编辑操作。
- ❑ 12.3 节简单讲解了如何在 Emacs 编辑器中快速移动窗口。
- ❑ 12.4 节介绍了 Emacs 编辑器中使用频率较高的常用技巧，包括撤销、恢复、复制、粘贴等。
- ❑ 12.5 节讲解了 Emacs 编辑器的几个高级功能，包括删除环、文本区域编辑、书签、灾难恢复功能。
- ❑ 12.6 节简单介绍了 Emacs 编辑器的目录模式及其中的常见操作。
- ❑ 12.7 节以发送、阅读电子邮件为例，介绍了 Emacs 编辑器中除文本编辑外的其他功能。

本书仅从简单的文本编辑角度出发，讨论了如何使用 Emacs 编辑器编辑文本，并未涉及其他更加高深的内容。感兴趣的读者可以参阅相关文档，了解更详细的使用说明。

第 13 章　Eclipse 编辑器

Eclipse 是近几年来兴起的一个开发平台。它最初是由 IBM 公司（International Business Machines Corporation，国际商业机器公司）开发的。Eclipse 的设计初衷是创建一个用于 Java 程序开发（Sun Microsystems 公司于 1995 年推出的一种跨平台程序设计语言）的 IDE 环境（Integrated Development Environment，中文含义为集成开发环境，简称为 IDE 环境），但随后它被扩展为可用于多种编程语言开发的 IDE 环境。

Eclipse 是一个开放源代码项目，人们在其基础上开发了许多插件。随着其功能、插件的开发，目前 Eclipse 已经扩展为可以使用任何语言的开发平台，甚至还可以进行图版绘制工作。虽然 Eclipse 经过了不断的扩展，但都有着统一的外观、相同的操作，这也是 Eclipse 能大行其道的原因之一。

为方便读者的后续学习，本书以 Java 为例介绍如何使用 Eclipse，涉及的知识点如下。

- 简单介绍 Eclipse 平台的诞生、发展及优势。
- Eclipse 的安装、启动、退出及基本界面介绍。
- 使用一个示例讲解如何使用 Eclipse 编辑器进行 Java 程序开发。

提示： 严格地讲 Eclipse 并不是一个编辑器，而是一个包含了编辑器的开发平台，因此初学者可以自行选择是否学习及使用。

13.1　Eclipse 开发平台概述与安装

Eclipse 是目前最为流行的 IDE 开发平台，目前使用的许多软件都是从 Eclipse 中开发而来的。现在 Eclipse 主要用于 Java 语言的开发，也可以用于 C、C++等高级语言开发，还可用于 Perl、Python、数据库等脚本语言开发。本节将简单介绍 Eclipse 开发平台的发展及其安装等内容。

13.1.1　Eclipse 平台概述

Eclipse 最初由 IBM 领导开发，其目标是用于替代商业开发环境 Visual Age for Java（由 IBM 公司开发的另一个商业化的 IDE 开发环境）。从 2001 年起，IBM 将 Eclipse 交由开源社区负责，现在 Eclipse 的管理工作是由非营利性软件供应商联盟的 Eclipse 基金会（Eclipse Foundation）完成的。

Eclipse 开发平台如同 Linux 操作系统一样，使用了一个非常小的核心，其他功能和组

件都通过插件的形式附加在该核心之上。使用 Eclipse 的插件开发环境，开发人员可以自由地开发 Eclipse 的插件。由于 Eclipse 中的插件并不仅仅局限于 Java 语言，因此开发人员可以自行开发用于任何语言并能无缝集成的开发环境。

对于开发人员而言，自由地修改开发工具及平台，使其获得了与 Eclipse 开发人员一样的权利。这大大方便了开发人员的开发工作，同时也极大地促进了 Eclipse 的发展。

Eclipse 的各个功能组件是由不同的计划完成的，其中部分计划及包含的具体内容如下。

- Eclipse：包括 Eclipse 的基础平台、客户端平台、Java 开发工具等。
- Eclipse Web 工具平台：主要用于 Web 开发的 Eclipse 平台，其中包括 HTML、CSS、JSP、JavaScript、SQL 等源代码编辑器，Web 服务向导，浏览器及测试工具等一整套 Web 开发工具的集合。
- Eclipse C/C++开发工具平台：使用 GCC 作为编译器的 C、C++集成化开发环境。
- Eclipse 测试和性能工具平台：一个可以让软件开发者对软件进行测试、调试等的平台，以及对软件进行性能等方面的测试的工具平台。

除此之外，Eclipse 还有许多计划，例如用于嵌入式开发的嵌入式客户端平台、用于图形化界面开发的可视化界面编辑器平台等，这些计划构成了整套的 Eclipse 开发平台。关于这些计划，本书中不再一一介绍，感兴趣的读者可以查阅相关资料。

本小节仅简单介绍 Eclipse 的基本情况，关于 Eclipse 的更多内容，读者可以查阅其官方网站进行了解：http://www.eclipse.org。

13.1.2　Eclipse 平台安装前的准备

安装 Eclipse 平台前，需要安装其工作的平台。由于 Eclipse 使用 Java 语言编写，并且安装包中并不包含 Java 运行环境，因此需要手动安装 JDK（Java Development Kit）环境。本小节将简单介绍 Eclipse 平台安装前的准备工作。

（1）下载 JDK 安装包

从 Sun 公司的官方网站上下载 JDK 安装包：Sun 公司的 Java 专栏地址为 http://java.sun.com/。读者可以使用任何自己熟悉的工具下载，例如图形化浏览器 Firefox、wget、links 浏览器等。

（2）解压 JDK 安装包

下载完成后，就可以安装 JDK 环境了。查看下载到的文件，这是一个名为 jdk-6u24-linux-i586-rpm.bin 的自解压安装文件。

提示：有许多软件的安装文件都以 bin 结尾，例如 Nvidia 官方（Nvidia，中文为英伟达，世界知名显示芯片制造公司）下载的显卡驱动等。这些软件都需要通过执行的方式安装，建议安装时先仔细阅读其安装说明，以免安装的软件出现问题。

可以通过执行文件的方式解压该文件：

```
#创建名为 jdk 的目录，并将自解压文件放入其中
# mkdir jdk
# mv jdk-6u24-linux-i586-rpm.bin ./jdk/
```

```
# cd jdk
#为自解压文件添加可执行权限，并执行该文件
# chmod  u+x jdk-6u24-linux-i586-rpm.bin
# ./jdk-6u24-linux-i586-rpm.bin
Unpacking...
Checksumming...
Extracting...
UnZipSFX 5.50 of 17 February 2002, by Info-ZIP (Zip-Bugs@lists.wku.edu).
  inflating: jdk-6u24-linux-i586.rpm
  inflating: sun-javadb-common-10.6.2-1.1.i386.rpm
  inflating: sun-javadb-core-10.6.2-1.1.i386.rpm
……
```

上面的命令中，先创建了一个新目录 jdk，并将下载到的安装文件移动到该目录中。为了运行该自解压文件，还必须为其添加执行权限。

提示：运行任何没有包含在环境变量 PATH 中的命令或文件时，都必须使用全路径。例如上面的这个例子中，由于当前目录并未被变量 PATH 所包含，因此使用 ./jdk-6u24-linux-i586-rpm.bin，这样表示运行当前目录下的 jdk-6u24-linux-i586-rpm.bin 文件。

（3）安装 JDK

解压完安装包后将会在当前目录下生成 7 个新文件：

```
# ls
jdk-6u24-linux-i586.rpm         sun-javadb-core-10.6.2-1.1.i386.rpm
jdk-6u24-linux-i586-rpm.bin     sun-javadb-demo-10.6.2-1.1.i386.rpm
sun-javadb-client-10.6.2-1.1.i386.rpm
                                sun-javadb-docs-10.6.2-1.1.i386.rpm
sun-javadb-common-10.6.2-1.1.i386.rpm
                                sun-javadb-javadoc-10.6.2-1.1.i386.rpm
```

这 7 个以 rpm 结尾的文件就是 JDK 的安装文件。

通过查看文件名或阅读说明文件，可以了解到只需要安装名为 jdk-6u24 的软件包即可。这时可以直接安装：

```
#使用 rpm 命令安装 jdk-6u24
# rpm -ivh jdk-6u24-linux-i586.rpm
Preparing...                ###########################################
[100%]
      package jdk-1.6.0_24-fcs.i586 is already installed
```

运行 rpm 命令安装软件包 jdk-6u24-linux-i586.rpm 之后，JDK 就安装完成了。

（4）设置环境变量

为了能正常使用 JDK，还必须为其设置相关环境变量。打开文件/etc/profile，在文件的最后加入以下内容：

```
#set java
JAVA_HOME=/usr/java/jdk1.6.0_24
CLASSPATH=.:$JAVA_HOME/lib.tools.jar
```

```
PATH=$JAVA_HOME/bin:$PATH
export JAVA_HOME CLASSPATH PATH
```

上面的内容中，以“#”开头的第 1 行为注释行，其作用为方便用户观看，也可以不必写入此行。

提示：在环境变量中的/usr/java/jdk1.6.0_24，可能会随安装的 JDK 版本不同而不同，因此在写入环境变量时，应该注意这个目录的具体名称。

（5）验证环境

确认写入正确的环境配置之后，可以通过重新登录系统或重新启动的方式，让环境变量生效。也可以使用以下命令：

```
#使用 source 命令让全局用户文件生效
# source /etc/profile
```

source 命令的作用是在当前 Shell 中重新运行/etc/profile 文件，以便让全局文件中的设置生效。关于此命令的用法将在第 16 章中介绍。

如果要验证 JDK 环境和环境变量是否都设置成功，可以运行命令 java 和 javac：

```
#使用 java 和 javac 命令验证
# java
Usage: java [-options] class [args...]
           (to execute a class)
   or  java [-options] -jar jarfile [args...]
           (to execute a jar file)
……
# javac
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
……
```

从上面两个命令的运行情况可以看出，相关环境变量配置正确。如果出现命令未找到的情况，就需要检查软件安装及环境配置是否正确。

13.1.3　安装 Eclipse 平台

在上一小节中，简单介绍了安装 Eclipse 平台前的准备工作。如果 JDK 的安装过程顺利，接下来就可以安装 Eclipse 平台。本小节将介绍如何安装并运行 Eclipse 平台。

1．下载Eclipse

可以通过 Eclipse 的官方网站下载 Eclipse 安装包：http://www.eclipse.org/。从其官方网站的下载频道可以看出，Eclipse 有多个不同版本，包括用于 Java 语言开发的 Eclipse IDE for Java Developers，用于 C、C++开发的 Eclipse IDE for C/C++ Developers，用于 PHP 语言的 Web 应用开发的 Eclipse for PHP Developers 等版本。本小节将以 Eclipse IDE for Java Developers 为例，讲解在 Linux 系统中如何安装 Eclipse 平台。

与前面的 JDK 环境一样，读者可以使用比较熟悉的工具下载 Eclipse。

2. 安装Eclipse

在本例中下载得到的文件名为 eclipse-java-helios-SR2-linux-gtk.tar.gz，从其文件名可以看出这是一个经过压缩的归档文件。

此时可以开始安装 Eclipse：

```
#将下载到的安装文件复制到目录/opt，并解压安装 Eclipse
# cp ~/eclipse-java-helios-SR2-linux-gtk.tar.gz /opt/
# tar -zxvf eclipse-java-helios-SR2-linux-gtk.tar.gz
eclipse/
eclipse/p2/
eclipse/p2/org.eclipse.equinox.p2.core/
eclipse/p2/org.eclipse.equinox.p2.core/cache/
……
```

上面的命令中，先使用 cp 命令将下载到的文件复制到目录/opt 中，然后使用 tar 命令恢复归档文件中的文件。

至此安装过程就已经完成了，查看安装的 Eclipse 如下：

```
#使用 ls 命令查看恢复的文件列表
# ls /opt/eclipse
about files     configuration   eclipse.ini     icon.xpm         p2
about.html      dropins         epl-v10.html    libcairo-swt.so  plugins
artifacts.xml   eclipse         features        notice.html      readme
```

上面的这些文件和目录就是安装后的 Eclipse，其中 eclipse 文件是一个可执行文件，运行这个文件就可以运行 Eclipse 平台。

3. 相关快捷设置

虽然已经安装完成，但是当我们要运行 Eclipse 时，还需要进入目录/opt/eclipse 并双击 eclipse 文件或使用命令/opt/eclipse/eclipse。通过这两种方式启动 Eclipse 都不方便，这时可以作一些设置以便于快速打开软件。

（1）添加快捷方式

由于 Eclipse 是一个在图形界面中运行的软件，因此最常见的设置是在桌面上添加一个 Eclipse 的图标，启动时只需要双击该图标即可。这种方式类似于 Windows 操作系统中的桌面快捷方式，为 Eclipse 建立桌面快捷方式：

```
#使用 ln 命令在用户桌面添加软链接
# ln -s /opt/eclipse/eclipse /root/Desktop/Eclipse
```

此时桌面上将会多出一个名为 Eclipse 的图标，双击该图标即可启动 Eclipse。

（2）命令启动

有时可能更希望使用命令的方式启动 Eclipse，这时可以新建一个名为 eclipse 的文本文件，在其中输入以下内容：

```
#!/bin/bash
/opt/eclipse/eclipse
```

然后执行以下操作：

```
#将新建的文本文件复制到目录/bin 中，然后添加可执行权限
# cp eclipse /bin/
# chmod +x /bin/eclipse
```

上面的命令中，先使用 cp 命令将文本文件 eclipse 复制到目录/bin 中，然后使用 chmod 命令为复制的文件添加执行权限。完成之后就可以在图形界面的控制台中输入命令 eclipse 启动 Eclipse 了。

注意：本小节中自制的文本文件 eclipse 是一个脚本文件，通过脚本的方式启动 Eclipse 时，还可以添加 Eclipse 的启动选项等。关于脚本文件的应用等内容，将在后续章节中详细介绍。

13.2　Eclipse 界面入门

由于 Eclipse 平台是一个集成化的开发环境，因此 Eclipse 的界面比较复杂。Eclipse 的界面复杂性不仅来自其自身拥有众多的窗口，不同的版本、使用的插件不同，Eclipse 的界面都可能会不同。因此在使用之前需要了解 Eclipse 的工作界面。本节将介绍 Eclipse 平台的界面等内容。

13.2.1　第 1 次启动

完成 Eclipse 的安装之后，就可以启动 Eclipse 平台了。首次启动时，还需要对 Eclipse 进行一些简单的设置。本小节将简单介绍 Eclipse 第 1 次启动的基本情况。

（1）进入图形界面

由于 Eclipse 是一个集成化的图形开发环境，因此只能在图形界面下工作。在启动 Eclipse 之前，需要将系统切换到图形环境。如果系统当前处于字符界面，可以使用命令 init 5 进入图形界面。

（2）设置工作区

第 1 次启动 Eclipse 时，会提示用户设置其工作区（workspace），如图 13.1 所示。

工作区负责保存和管理用户的资源，这些资源以项目名为目录名存放在工作区内，其中还包含了项目的相关档案、数据文件等内容。Eclipse 默认的工作区位于目录/root/workspace 中，如果需要修改工作区的位置，只需要在此时输入工作区的目录即可。

（3）启动界面

设置完 Eclipse 平台的工作区之后，Eclipse 就会启动了。首次启动的欢迎界面如图 13.2 所示。

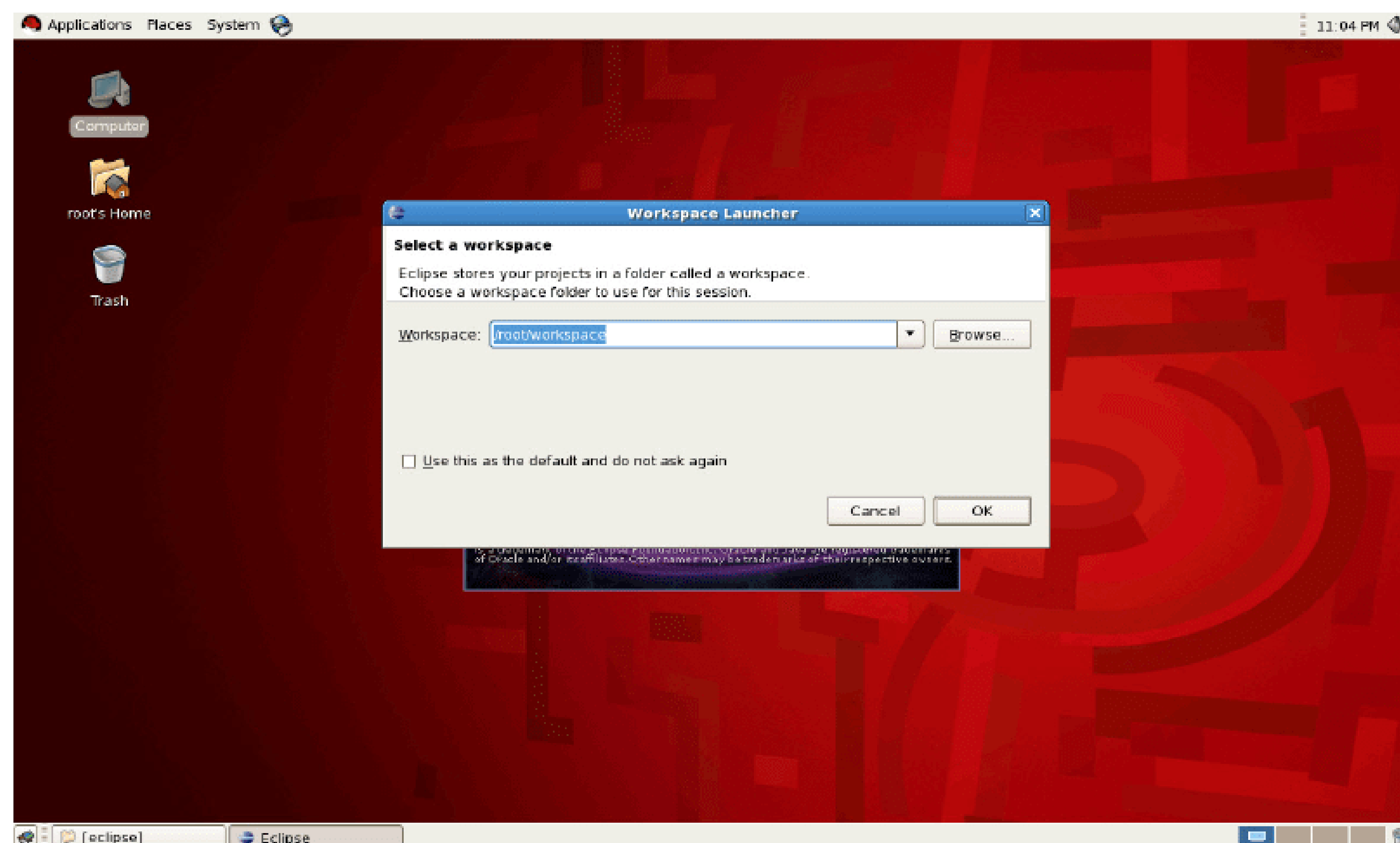


图 13.1　设置 Eclipse 的工作区

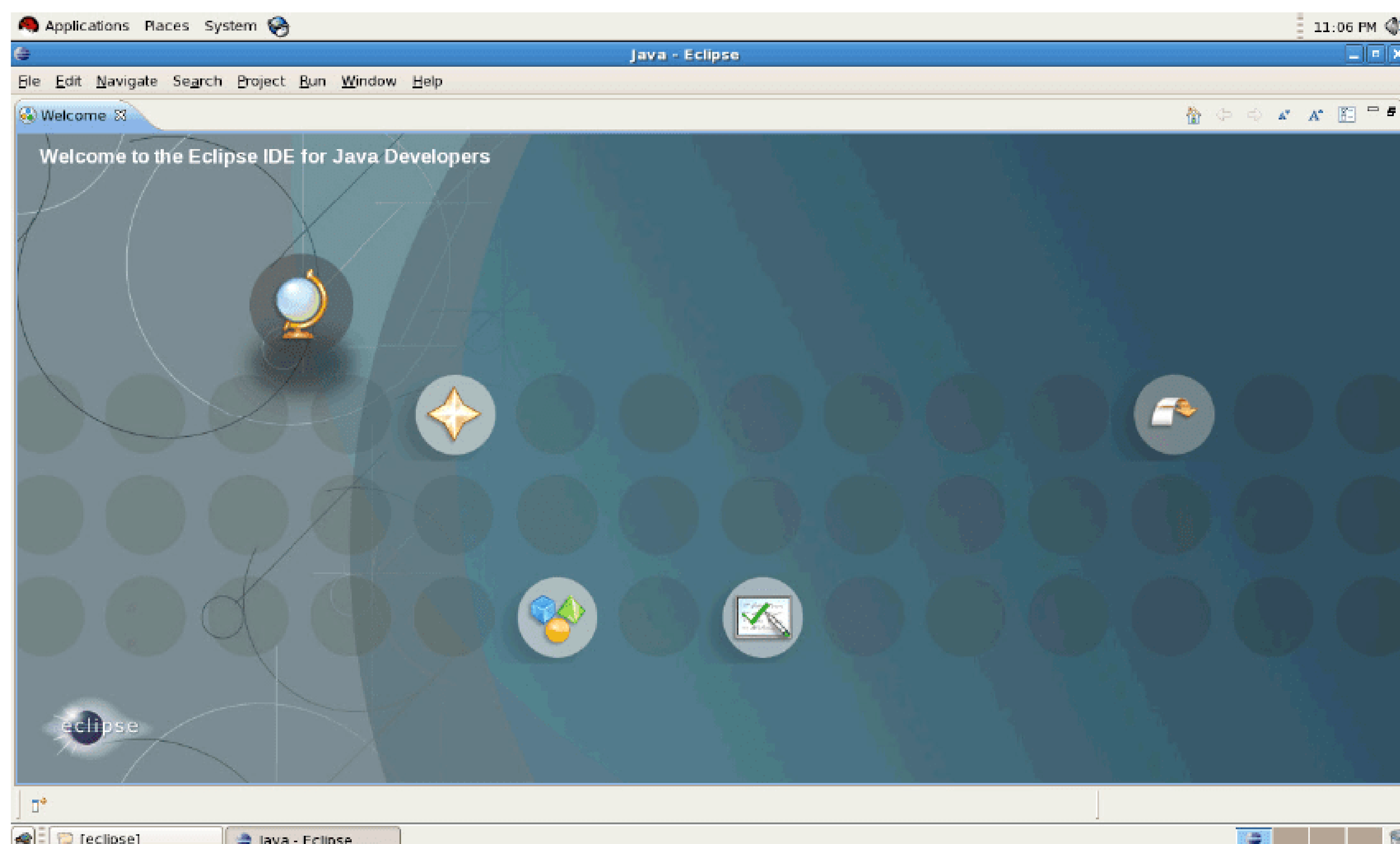


图 13.2　Eclipse 首次启动的欢迎界面

在这个首次启动的欢迎界面中，用户可以点击相关图标获取帮助、示例、教程等内容，关于这些内容本书不做介绍，感兴趣的读者可以自行阅读并参考这些内容。

13.2.2　Eclipse 界面介绍

Eclipse 提供了一个图形界面中的编程接口，这个图形界面中的编程接口通常称为工作台，如图 13.3 所示。

Eclipse 的工作台非常像一个应用程序的界面，这就是 Eclipse 的编程接口。在这个图形接口中，用户可以查找项目、档案资源等内容。工作台的主要工作是管理项目及其中的档案资源。

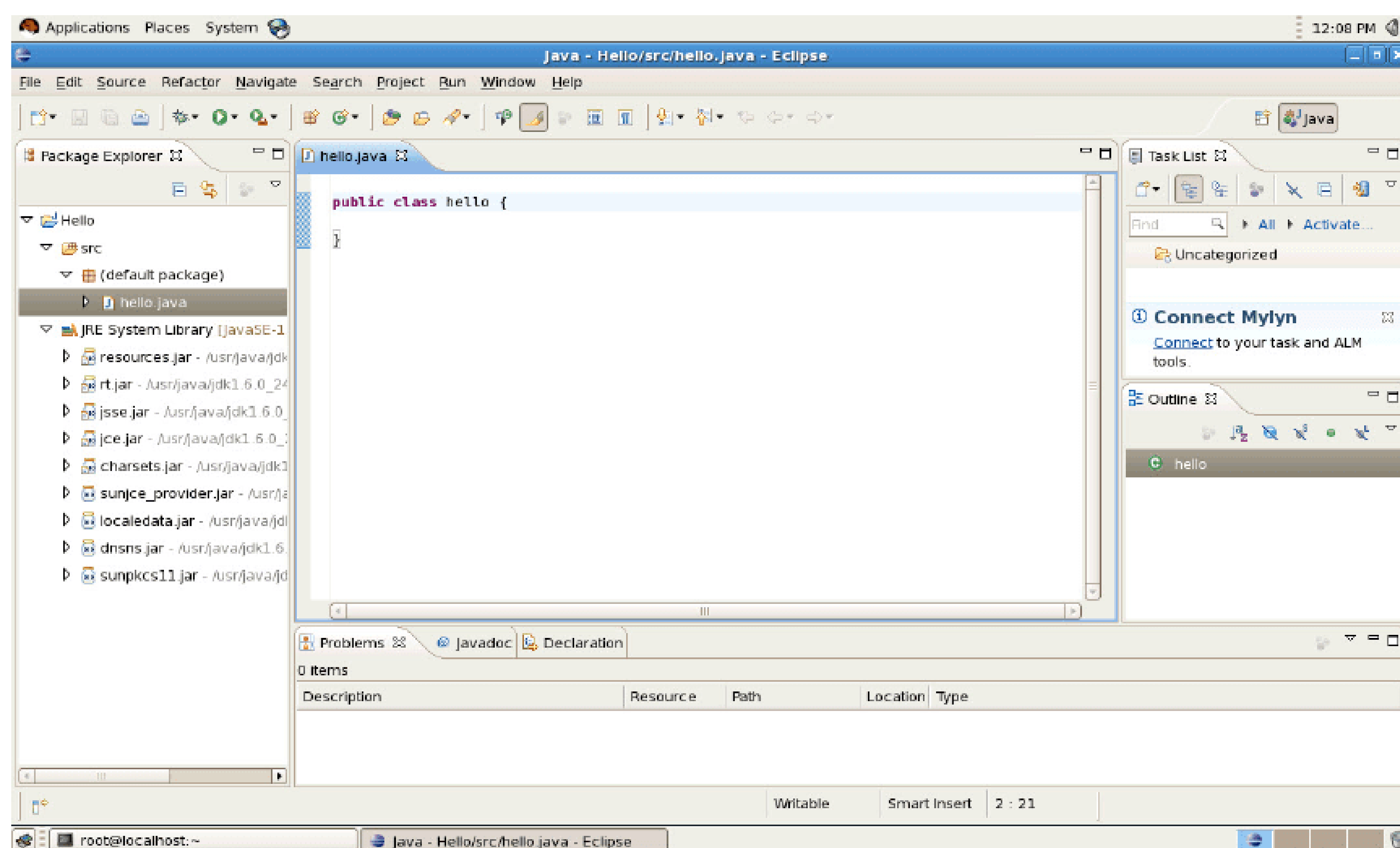


图 13.3　Eclipse 工作台

提示：与 Windows 系统中的集成化编辑环境不同，工作台并不会执行代码，也不会为代码排除错误。

如同我们经常看到的应用程序一样，Eclipse 的工作台包含了标题栏、菜单栏、工具栏和底部的状态栏等。Eclipse 的工作台包含了许多小窗口，从这些窗口中用户可以从不同的角度查看已存在的项目。例如在 Outline 窗口中，用户可以看到项目中的类的基本概况等。

在 Eclipse 工作台的正中，是一个文本编辑器，用户可以通过此文本编辑器编辑代码。在工作台的文本编辑器中，用户可以打开多个文本编辑器窗口，同时编辑多个文本。

13.2.3　操作窗口

许多时候可能需要在当前工作台中查看、新建其他的窗口，或者需要重叠、关闭窗口。例如需要从其他角度查看项目时，都涉及窗口的操作。本小节将简单介绍 Eclipse 工作台中的窗口的操作方法。

（1）新建窗口

要打开一个新的窗口，在菜单栏中单击 Windows，在弹出的下拉菜单中单击 Show View，然后选择要打开的窗口即可，如图 13.4 所示。

（2）查看重叠的窗口

有时打开的窗口可能会出现多个窗口重叠的情况。与 Windows 中一样，要查看处于后台的窗口，可以单击窗口标签查看窗口。

（3）重叠窗口

有时可能打开的窗口较多，需要重合窗口，即重叠窗口。此时可以单击窗口标签并拖动到指定窗口标签，此时这些窗口将会重叠。

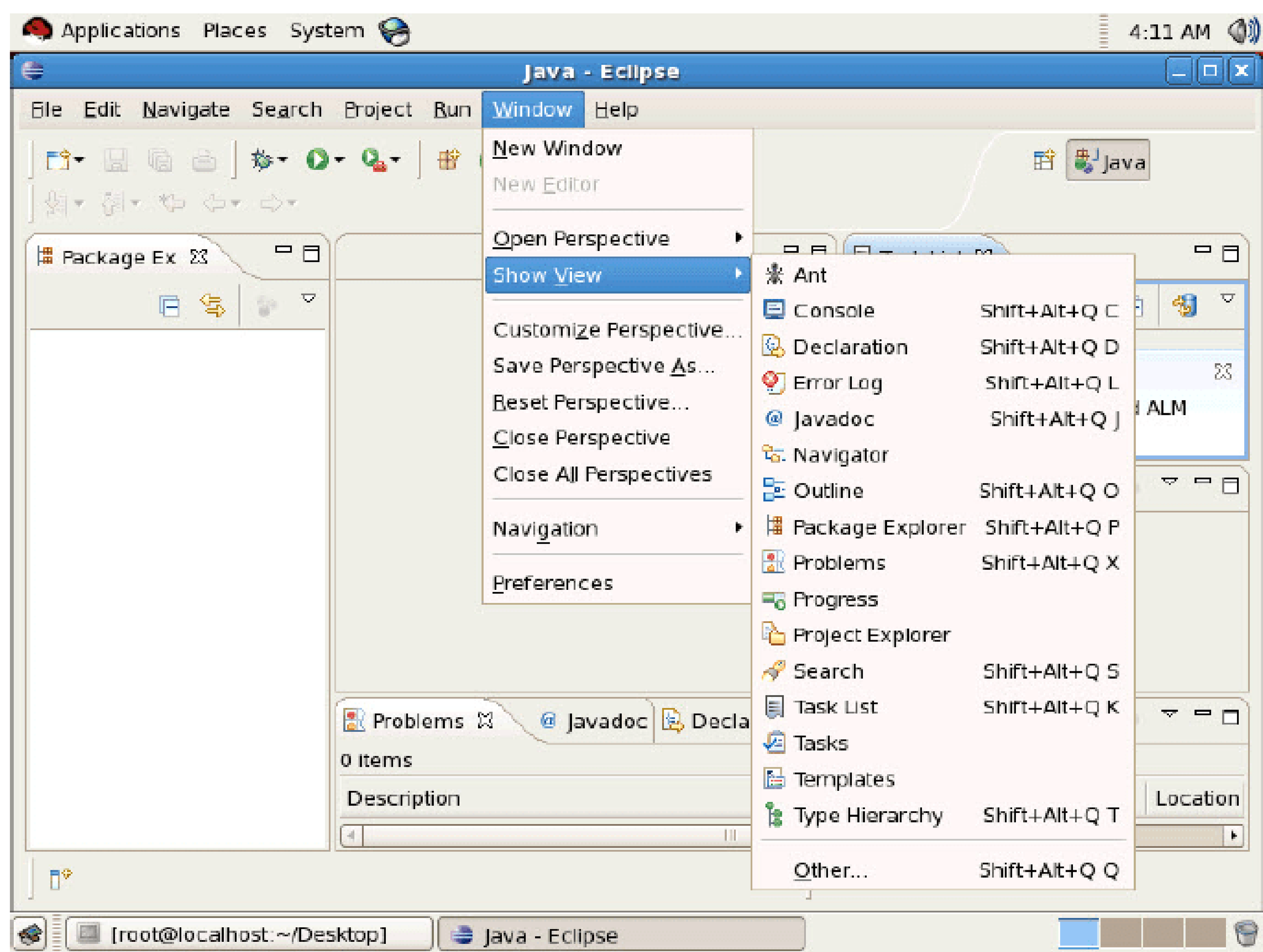


图 13.4　添加窗口

（4）关闭窗口

如果需要关闭一个窗口，可以单击相关窗口标签，单击右键弹出快捷菜单，如图 13.5 所示。在快捷菜单中选择 Close 即可关闭该窗口。

（5）重排窗口

有时在工作台中打开了许多个窗口，这些窗口可能会影响用户的整体操作，此时可以通过单击菜单 Windows 中的 Reset Perspective。重新排布所有窗口，需要注意的是重排所有窗口后，工作台将会恢复到初始状态，用户新打开的窗口将会关闭。

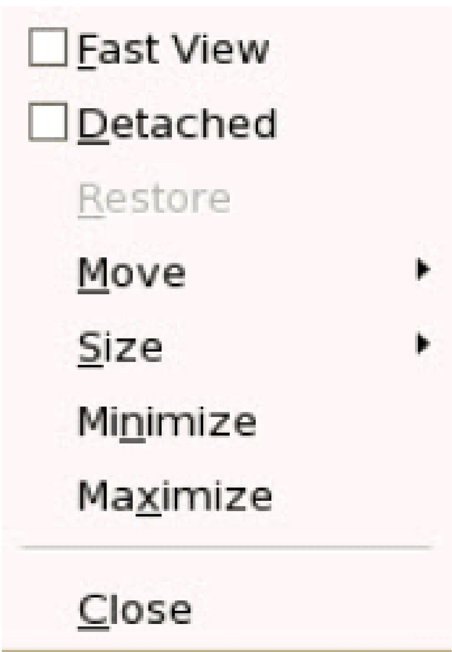


图 13.5　窗口的右键快捷菜单

（6）最大化、最小化窗口

有时可能需要查看的窗口中有许多内容，默认的窗口大小看上去非常不方便，此时可以在窗口的右键快捷菜单中，单击 Maximze 将窗口最大化。最大化窗口会占满整个工作台，其他窗口都会最小化至两侧，如图 13.6 所示。

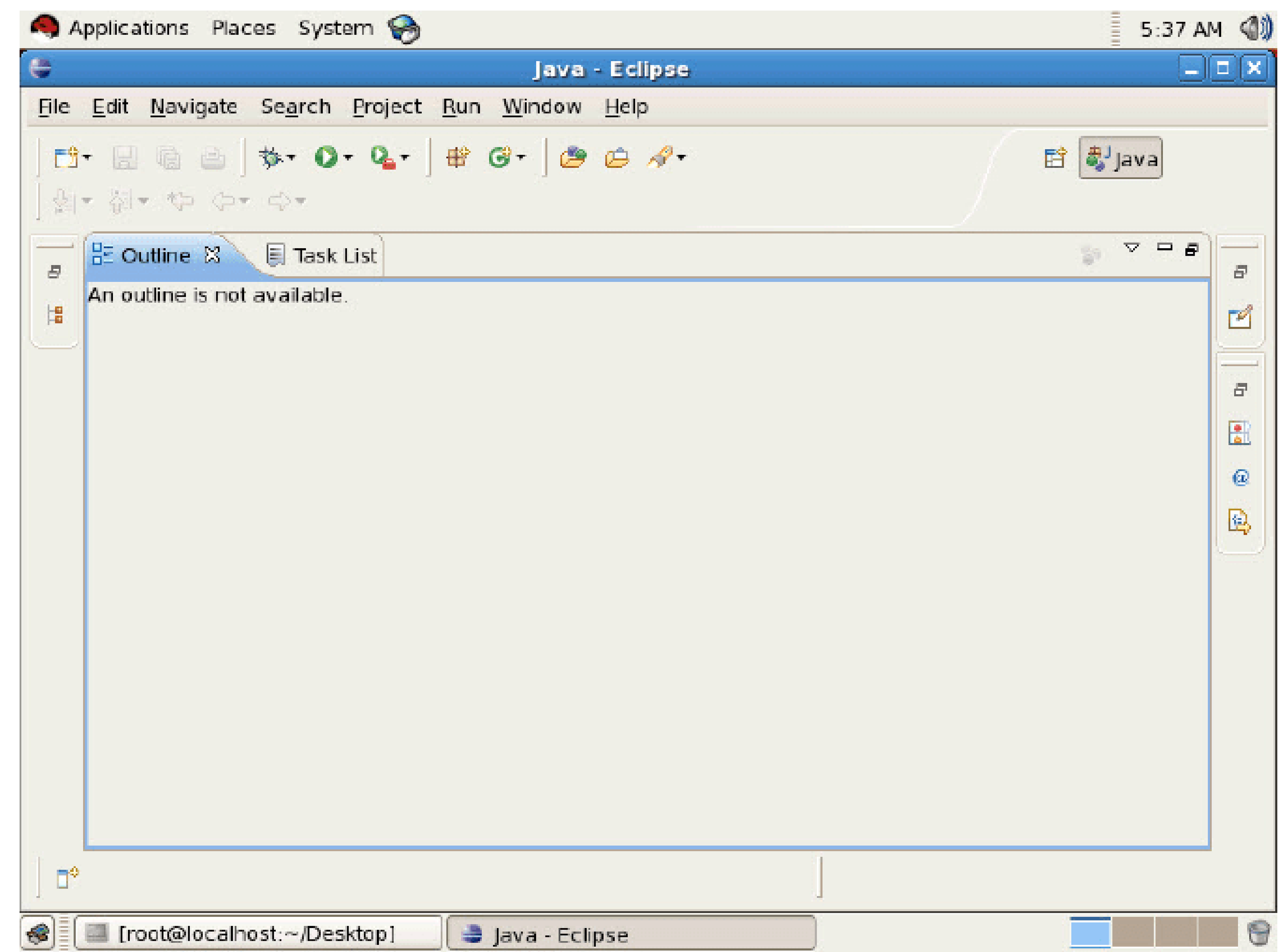


图 13.6　窗口最大化

从图 13.6 中可以看出，其他窗口都被最小化至工作台的两侧。如果需要恢复窗口，可以单击最小化的窗口面板顶端的 Restore 按钮。如果要将整个窗口恢复，可以单击已最大化的窗口标签最右侧的 Restore 按钮。

如果不需要查看某个窗口，可以将这个窗口最小化，以免影响查看其他的窗口。要最小化窗口，可以在相应窗口的右键快捷菜单中单击 Minimize 选项，要恢复窗口，同样可以单击小面板顶端的 Restore 按钮。

（7）移动窗口

有时需要移动工作台中的某一个窗口，这时单击窗口标签，通过拖动的方式移动窗口到目标位置即可。

由于窗口操作在实际使用过程中操作比较频繁，因此本小节简单介绍了 Eclipse 中的窗口操作。除本小节中介绍的窗口操作之外，还有许多其他操作，读者可以通过阅读相关帮助信息或相关文档了解具体内容，此处不再赘述。

13.3　使用 Eclipse 开发 Java 程序

Eclipse 中自带了许多便于用户输入代码的工具，利用这些工具不仅可以加快代码的输入，还可以降低编写代码的难度等。本节将简单介绍如何利用 Eclipse 开发 Java 程序。

13.3.1　建立编程项目

使用 Eclipse 集成化开发环境编程时，首先要创建一个编程项目。创建编程项目时需要为编程项目取一个合适的名称，通常都按编程项目的功能对项目命名。

建立 Java 编程项目，可以依次单击菜单栏中的 File→New→Java Project，此时会弹出建立 Java 项目对话框，如图 13.7 所示。

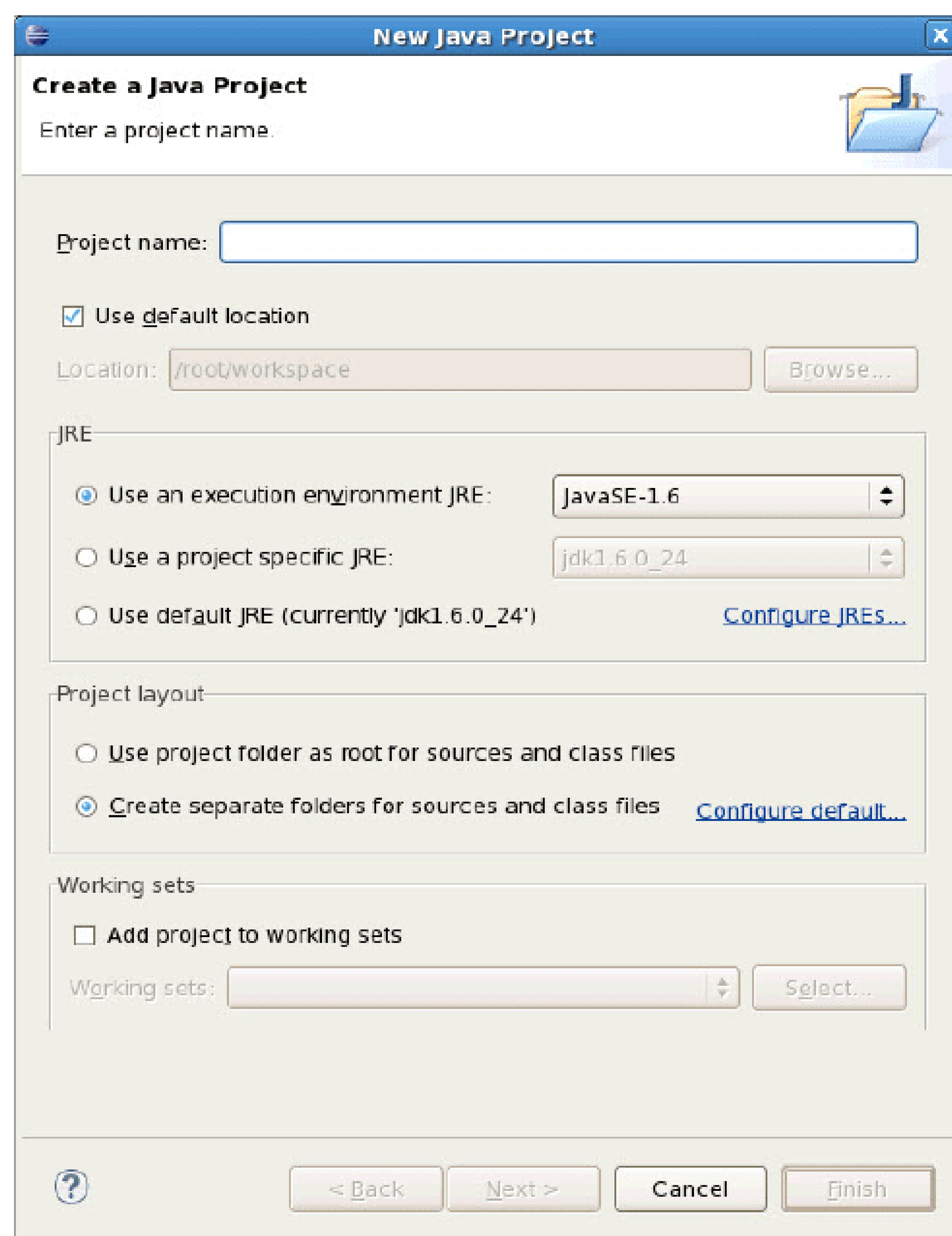


图 13.7　建立 Java 编程项目对话框

（1）默认情况下，Eclipse 会将项目的文件放在工作区中。如果需要指定项目相关文件存放目录，可以取消选择 Use Default location 选项，然后在 Location 文本框中输入指定的目录，或单击 Browse 按钮选择目录。

（2）在图 13.7 中， Eclipse 将使用 JavaSE-1.6 作为其 JRE 环境（Java Runtime Environment，Java 运行环境）。如果要改变这个默认设置，可以在 Use an execution environment JRE 后的下拉选择框中选择环境。如果要使用特殊的 JRE，可以在 Use a project specific JRE 后的下拉列表中选择，也可以选中 Use default JRE，使用系统中已经安装的其他 JRE 环境。

（3）在建立编程项目时，还可以自定义其他选项。例如在 Project layout 标签中指定文件设置，在 Working sets 中设置工作集等。通常这些选项按实际需要设置即可。

（4）本例中，用户在 Project name 中输入 Hello World 作为编程项目名称，其他设置均使用默认值，然后单击 Finish 按钮完成创建。

（5）完成创建之后，Eclipse 会使用项目名称在工作区中建立一个目录，并在该目录中创建两个名为 bin 和 src 的子目录，分别用于存放输出的文件和源代码。

13.3.2　建立 Java 类

Java 是一种面向对象的程序语言，因此在完成 Java 编程项目的创建之后，必须要创建类。本小节将使用一个简单的实例介绍如何在 Eclipse 中创建 Java 类。

小知识：面向对象是目前广为使用的一种程序设计方法，普遍认为面向对象是在程序设计过程中使用封闭、继承、多态的编程手法。目前可以使用的面向对象编程语言有 Java、C++、C#等，关于此部分的更多内容，读者可以参考相关书籍。

创建 Java 类可以通过在菜单栏中依次单击 File→New→Class，此时将会弹出 Java 类创建对话框，如图 13.8 所示。

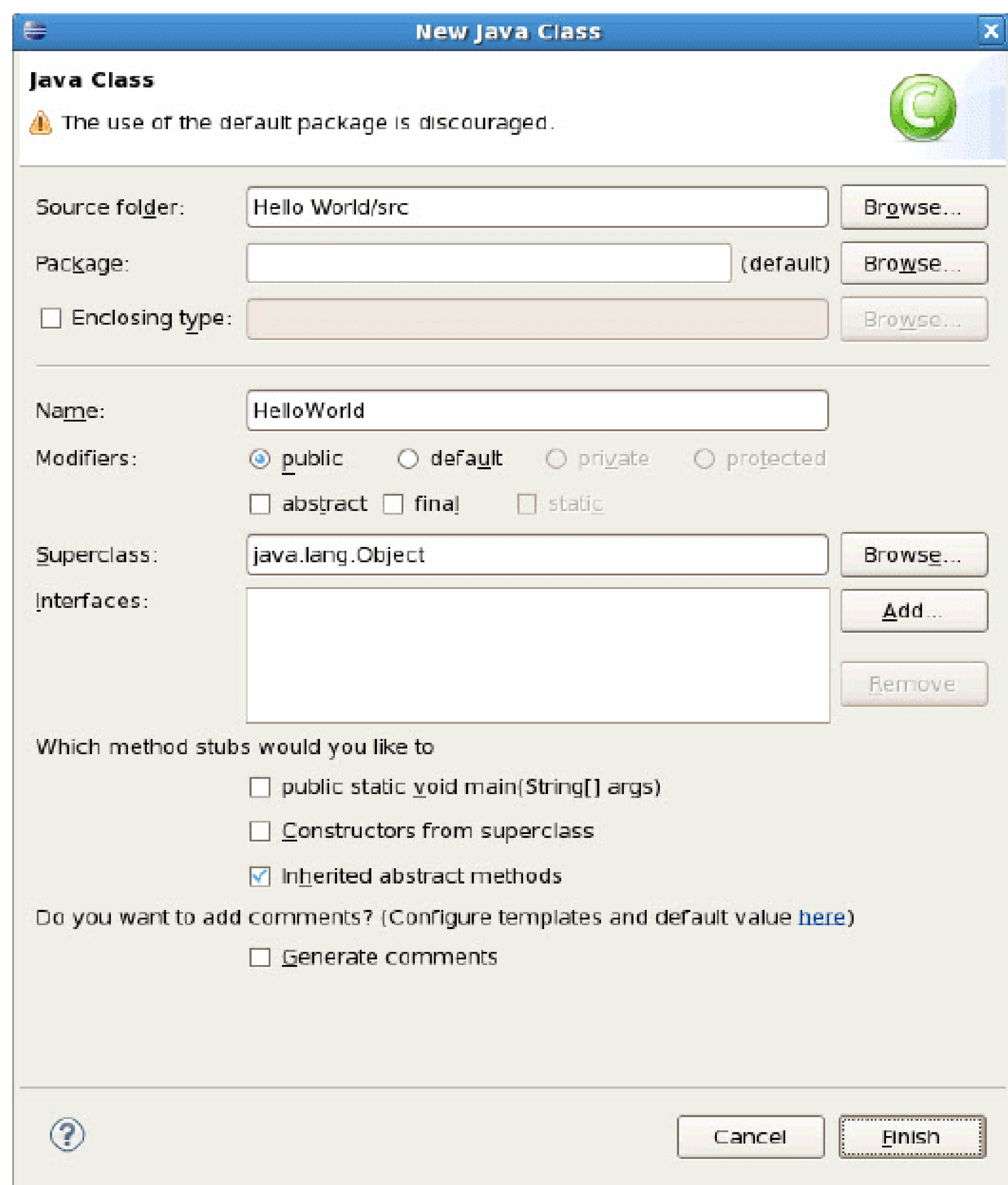


图 13.8　Java 类创建对话框

在上面的 Java 类创建对话框中，Source folder、Package 为类文件存放的位置和包，通常这两个选项不需要修改，保持默认值即可。Name 选项框中是需要用户输入的类名，Modifiers 选项后面是创建类的类型。除此之外，还可以选择 Superclass 超类，添加接口 Interfaces，为类添加执行入口“Public static void man(String[] args)”等。

在本例中使用的类名为 HelloWorld，类型为 Public。其他设置均保持默认值，输入完成之后，单击 Finish 按钮即可创建类。创建完成后，Eclipse 将会在项目目录的 src 子目录中创建类的源文件，并使用编辑器打开，如图 13.9 所示。

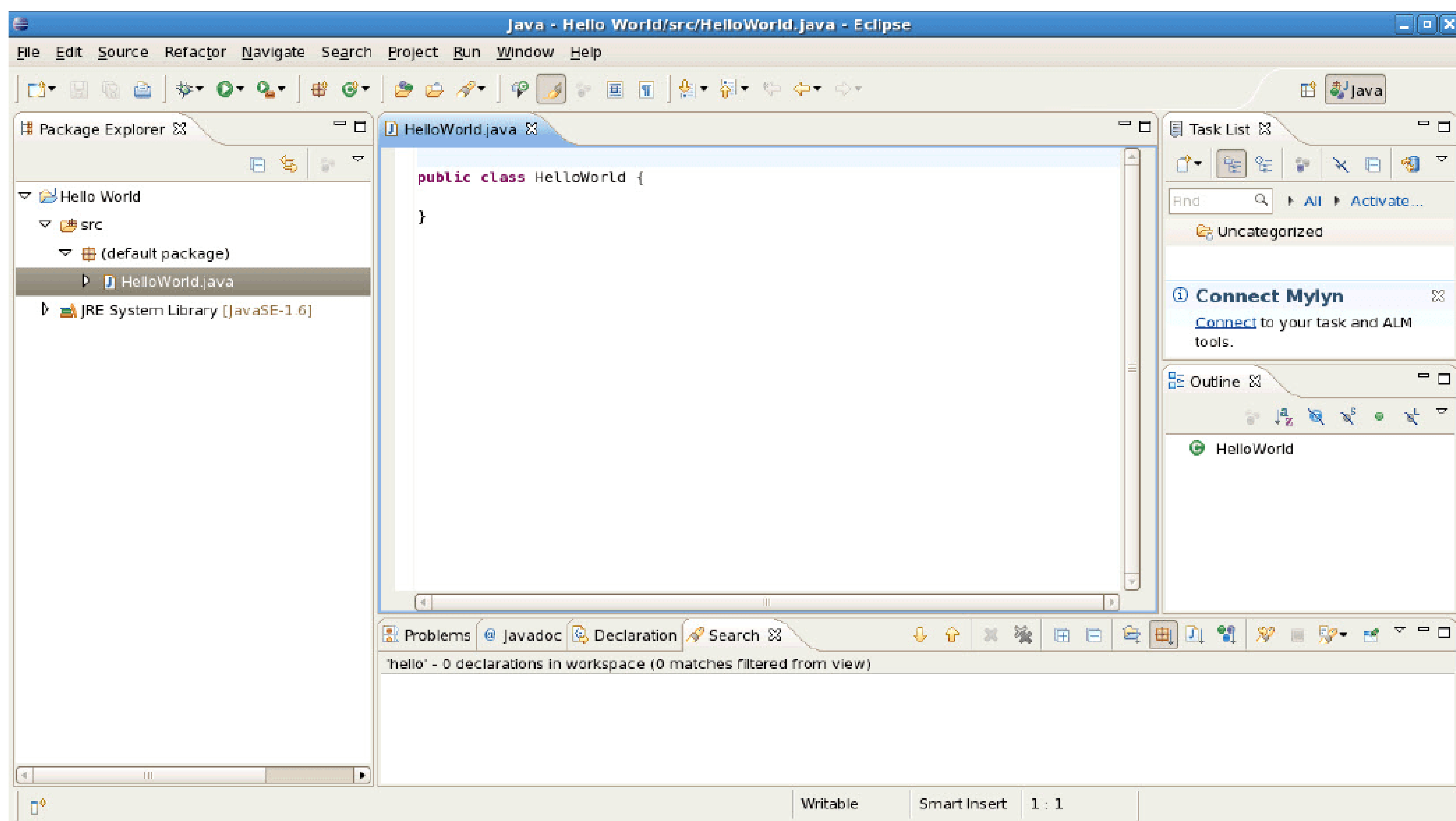


图 13.9　自动创建并定义类

从图 13.9 中可以看出，创建的类文件位于项目目录中的 src 子目录中（左侧的 Package Explorer 窗口中），并自动将定义类的语句写入文件中（编辑器中的文本）。

13.3.3　输入编程内容

项目的类创建完成后，就可以写入类的相关功能性语句块了。本小节将简单介绍在 Eclipse 中编写代码的技巧。

（1）代码自动完成功能

Eclipse 提供了代码自动完成功能：当用户输入左括号“（”时，Eclipse 将会自动补全右括号“）”；用户输入左大括号“{”并按下 Enter 键时，Eclipse 工作台中的编辑器将自动在光标所在行的下一行补全右大括号“}”。

（2）代码自动缩进

当用户在类中输入内容并换行时，Eclipse 工作台中的编辑器会自动为输入的内容使用缩进，以便于用户阅读，如图 13.10 所示。当用户输入“main(String args[]) {”并按下 Enter 键时，Eclipse 的编辑器会自动缩进（即图 13.10 中的 System.out.println(“Hello World!”);这

一行的缩进）。如果继续按下 Enter 键，编辑器依然会使用缩进。

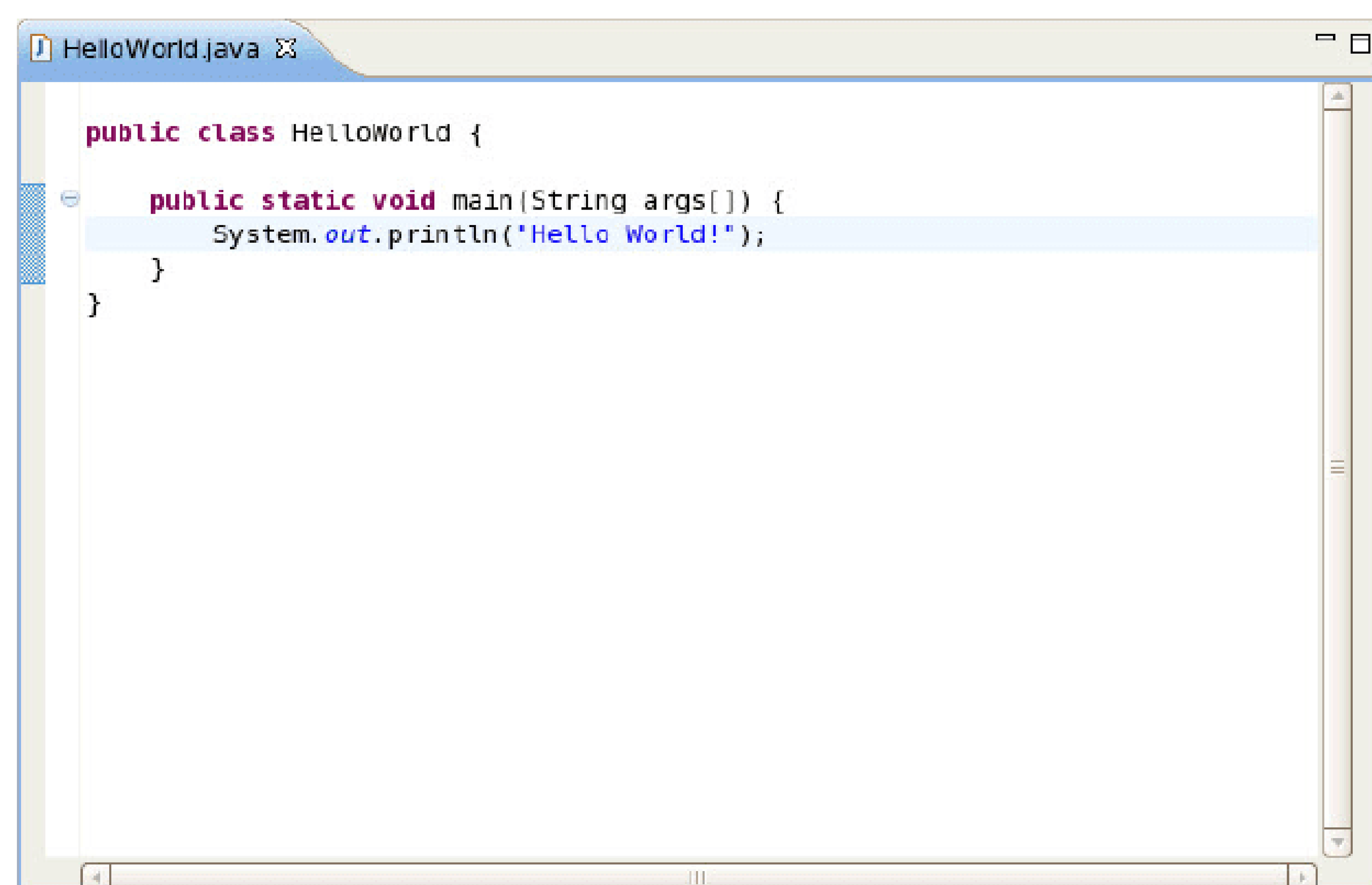


图 13.10　代码自动缩进

（3）代码提示功能

Eclipse 编写代码时，还可以使用其自带的代码提示功能。使用代码提示功能可以快速地输入代码，也可以查看方法的用法等。

例如在执行入口方法中输入“System.”，等待几秒钟就会弹出代码提示窗口，如果选中代码提示窗口中的某项内容，还会显示选中方法的注解，如图 13.11 所示。

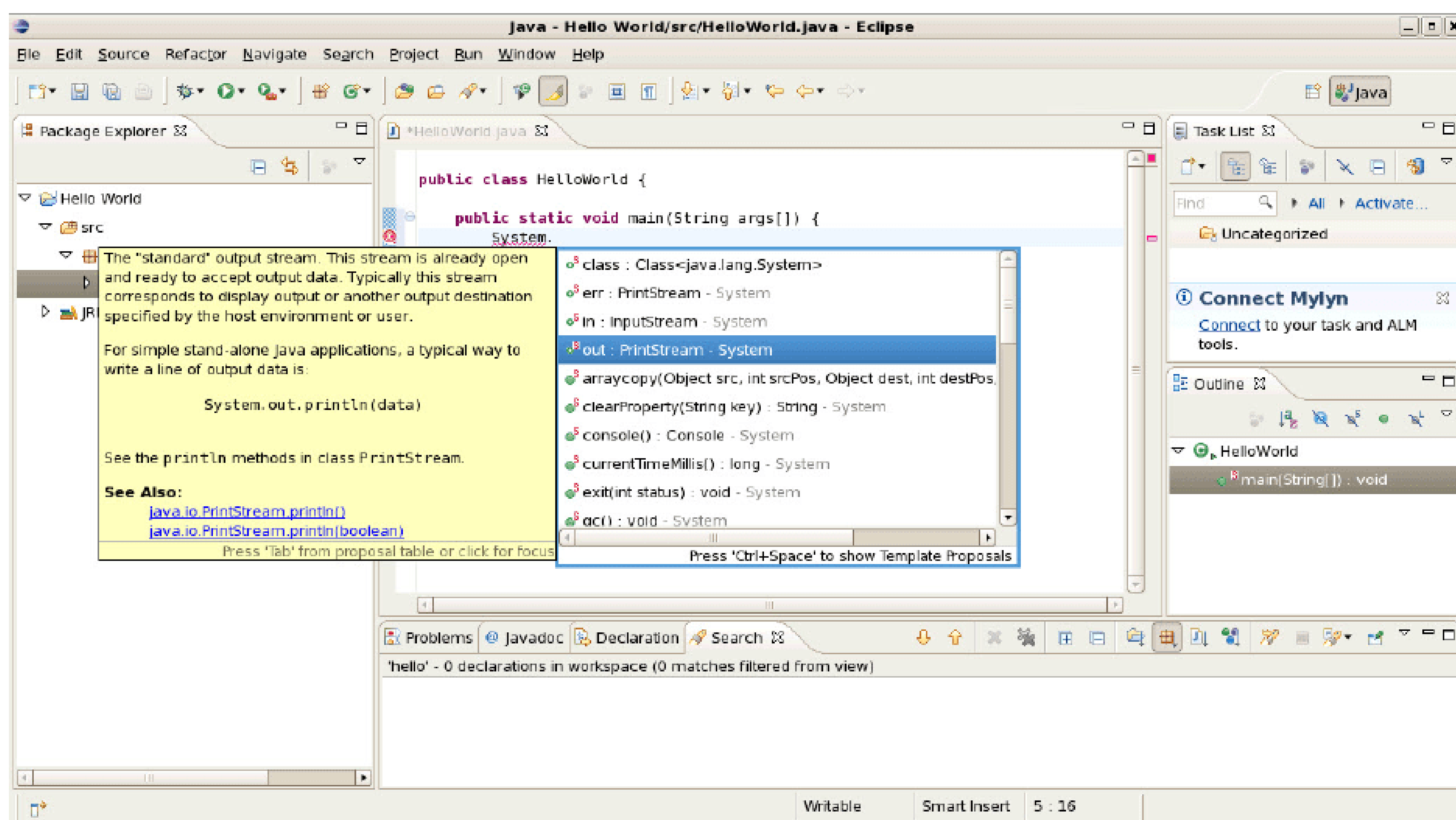


图 13.11　代码自动提示功能

如果当前选中项就是要输入的内容，可以按下 Enter 键，Eclipse 会将选中的内容自动输入到编辑器中。这时如果继续输入，当输入至具体方法时，Eclipse 还会提示方法的使用说明，如图 13.12 所示。

不熟悉方法的程序员，可以通过使用说明选择需要使用的方法。

（4）错误提示

当程序员输入的代码出现错误时，Eclipse 会使用红色的波浪线提示用户。此时如果用户将鼠标移动到波浪线之上，Eclipse 将会显示修正意见，如图 13.13 所示。用户按需要选

择修改意见中的选项，即可完成错误修改工作。

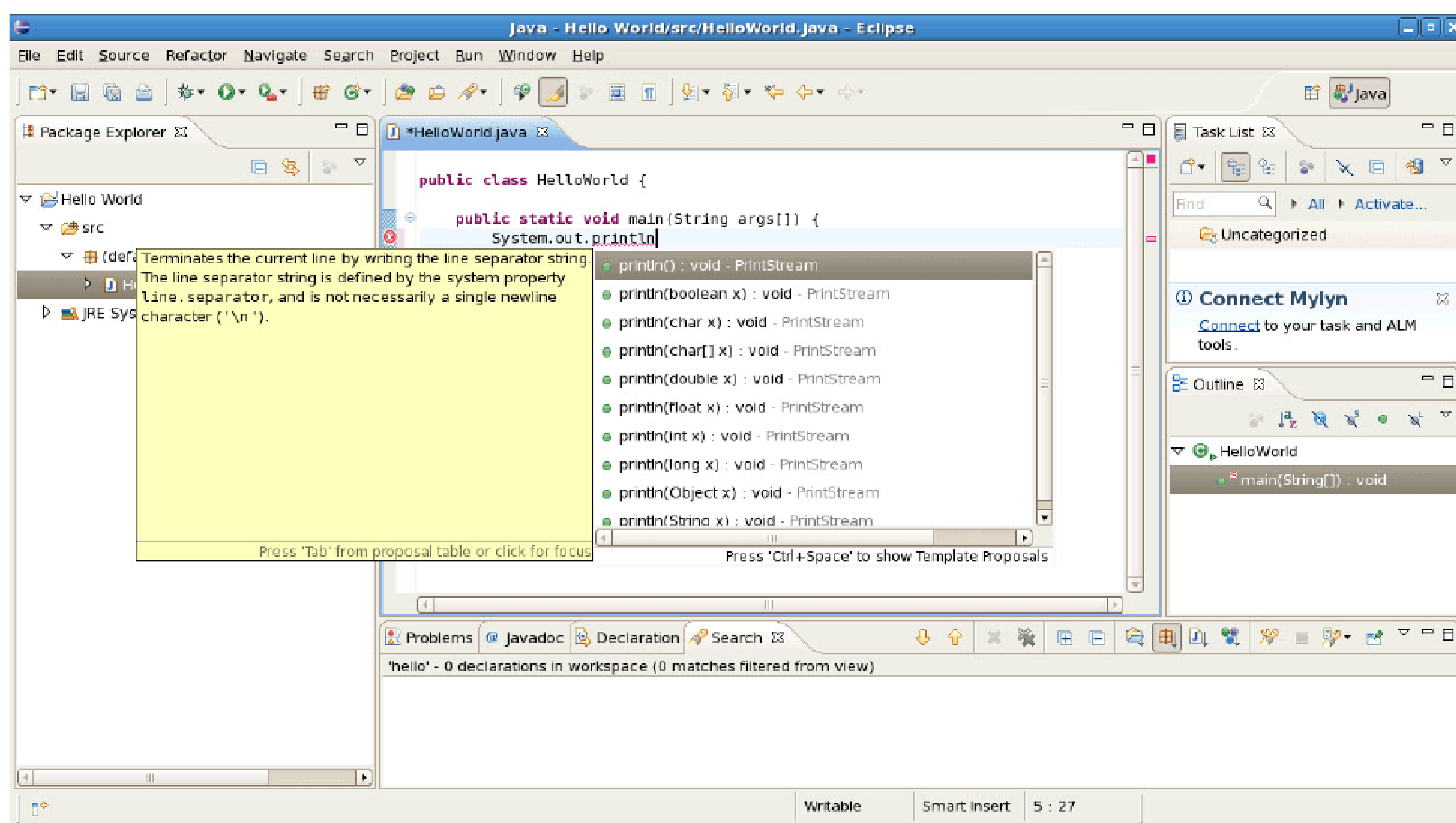


图 13.12　提示方法的使用说明

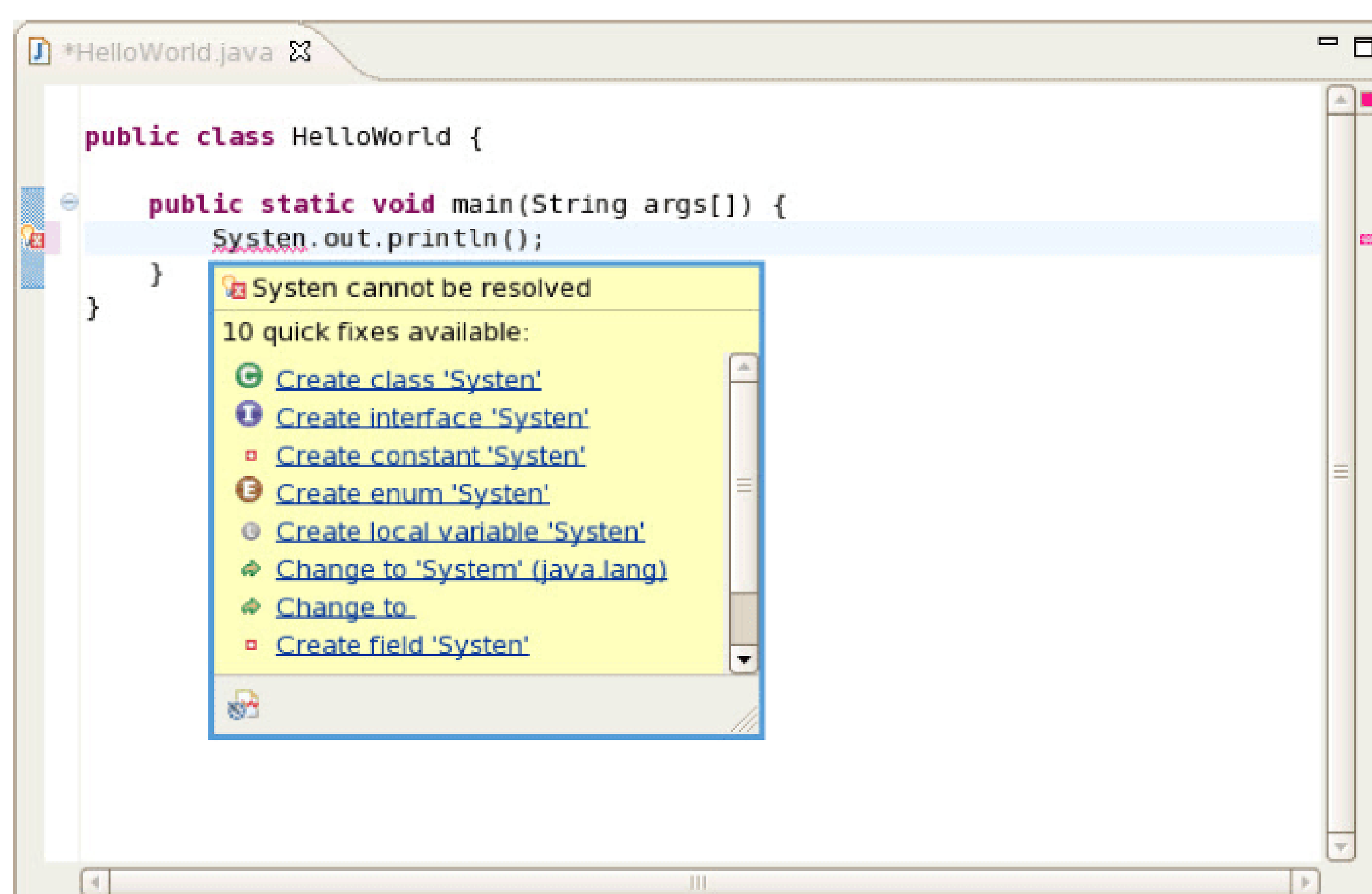


图 13.13　错误修正意见

合理地使用 Eclipse 工作台提供的各种功能，可以大大提高程序员输入代码的速度，也能从一定程度上提高代码的质量。

13.3.4　运行 Java 程序

如果需要运行输入的代码，可以依次单击菜单栏的 Run→Run。这时如果当前编辑器中还有未保存的文档，系统将会提示用户保存。完成文档的保存之后，Eclipse 会自动对用户编写的代码进行编译，并将编译之后生成的文档放入项目目录中的 bin 子目录中。

完成保存编译工作之后，Eclipse 会打开一个名为 Console 的窗口，并在其中显示程序

运行的结果，如图 13.14 所示。

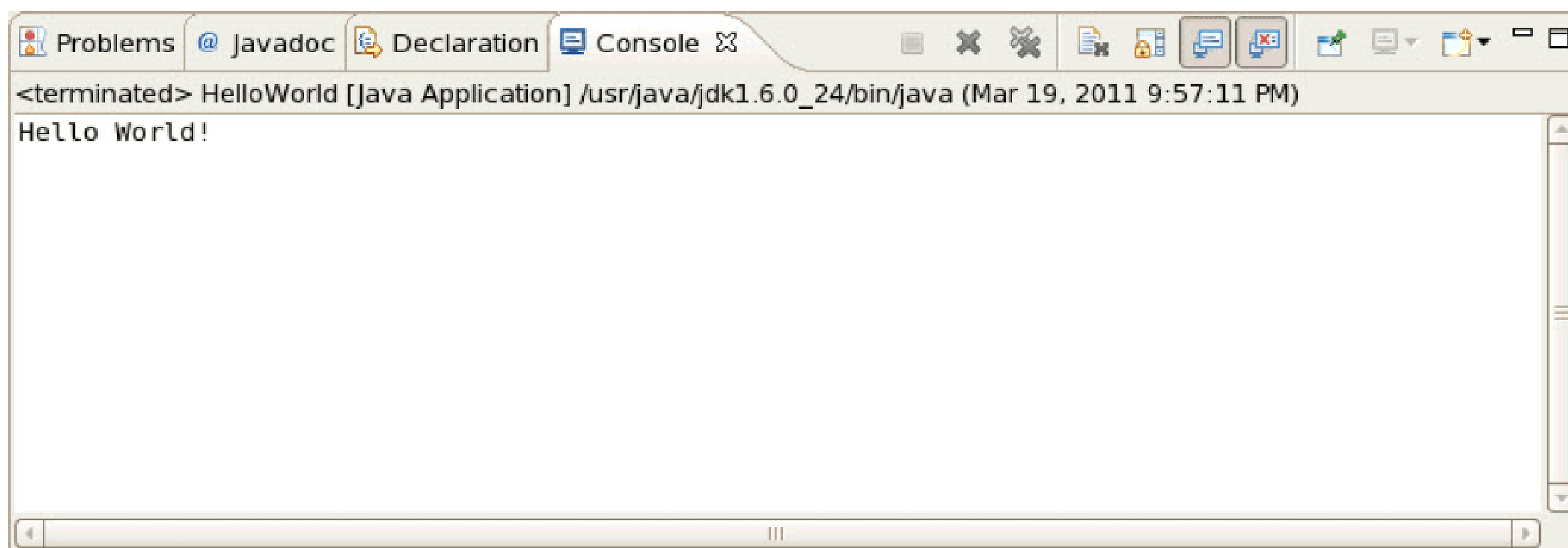


图 13.14　Console 窗口

13.3.5　调试程序

有时可能编写的程序没有达到预期的效果，这可能是程序代码中存在错误造成的。代码中常见的错误可以分为如下两类。

- 一种是由于程序不小心输入错误造成的语法错误，这种错误一般可以通过 Eclipse 自带的错误提示修正。
- 另一种是由于程序员编写程序时产生的逻辑错误，由于 Eclipse 通常不会提示逻辑错误，因此需要对程序进行调试。调试就是一步一步地查看程序的执行过程，从其表达式和变量的值中查找程序中的 Bug。

为讲解调试程序的步骤，此处引入一个小程序，这个小程序用于计算从 1 到 10 这 10 个整数的和，代码入下：

```
public class Test {

    /**
     * @param args
     */
    public static void main(String[] args) {
        int sum=0;
        int a=1;
        while(a<10){
            sum+=++a;
        }
        System.out.println("sum="+sum);
    }

}
```

注意：在 Java 语言中，通常使用“/*”表示注释的起始位置，“*/”表示注释的结束位置。

为了捕捉到程序设计过程中的异常，通常需要在代码中的关键语句附近添加断点，然后通过程序执行到断点时各表达式和变量的值，判断程序设计是否存在问题。

1. 为关键句添加断点

断点是程序在调试过程中的暂停点，即当程序执行到断点所在位置时会暂停（并不会执行断点所在的语句），只有用户允许继续执行之后才能继续执行。为哪些语句设置断点非常重要，通常关键点位于以下语句附近。

- ❑ 条件判断语句。查看条件判断语句之前各变量和表达式的值，可以了解程序的后续执行过程。
- ❑ 流程控制语句。一个未知的 Bug 可能会改变流程控制语句的执行顺序，从而改变整个程序的执行。
- ❑ 复杂的赋值语句。输入代码时的失误，可能会导致这些复杂的赋值语句没有达到预期的目的。

除此之外，一些方法的返回值等也是需要关注的对象。

添加断点时，应该先将光标移动到需要添加断点的行，然后依次单击菜单栏的 Run→Toggle Line Breakpoint 添加断点。Eclipse 会自动标记断点，如图 13.15 所示。

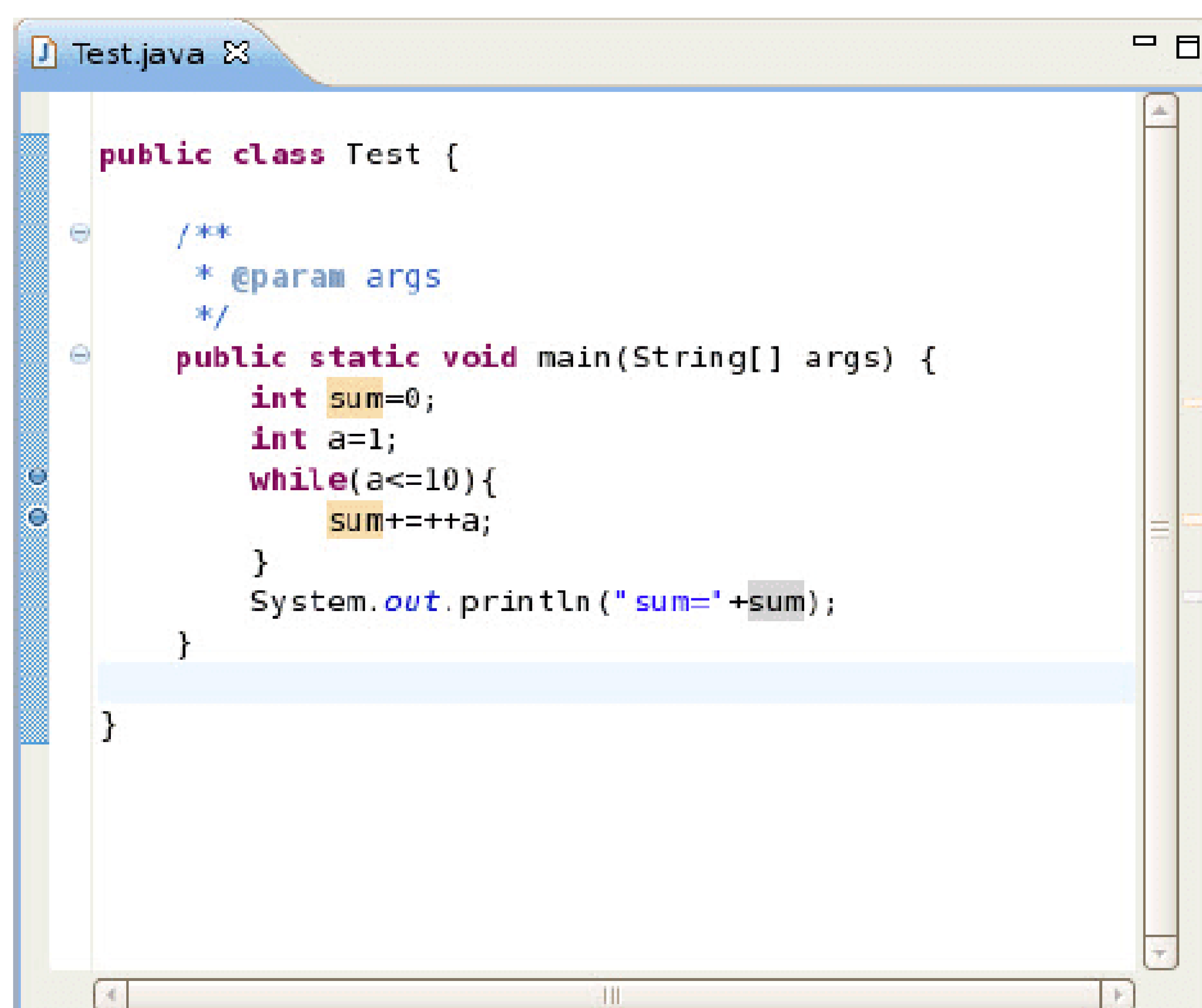


图 13.15　添加断点

在上面这个例子中，添加的两个断点分别位于代码“while”和“sum+=++a”这两行，编辑器使用了两个小圆点表示此行添加了断点。

2. 调试代码

添加断点之后就可以开始调试代码了，依次单击菜单栏的 Run→Debug，或按下 F11 键，此时 Eclipse 将进入 Debug 视图，如图 13.16 所示。从图中可以看出，当前程序已开始执行，并且停留在第 1 个断点处（图 13.16 编辑器中的小箭头所示），在 Variables 窗口中可以查看到当前断点各变量的值。如果程序在调试过程中有输出，可以从 Console 窗口中查看。

通过查看各变量值的方式，可以判断当前运行状态是否与预期值相符。如果不符，则要查看代码是否有错误，相反就需要继续调试。要继续调试，可以依次单击菜单栏中的 Run→Step over，越过当前断点。此时程序会执行到下一个断点处暂停，查看 Variables 窗口中

变量的值是否正确，依此类推。

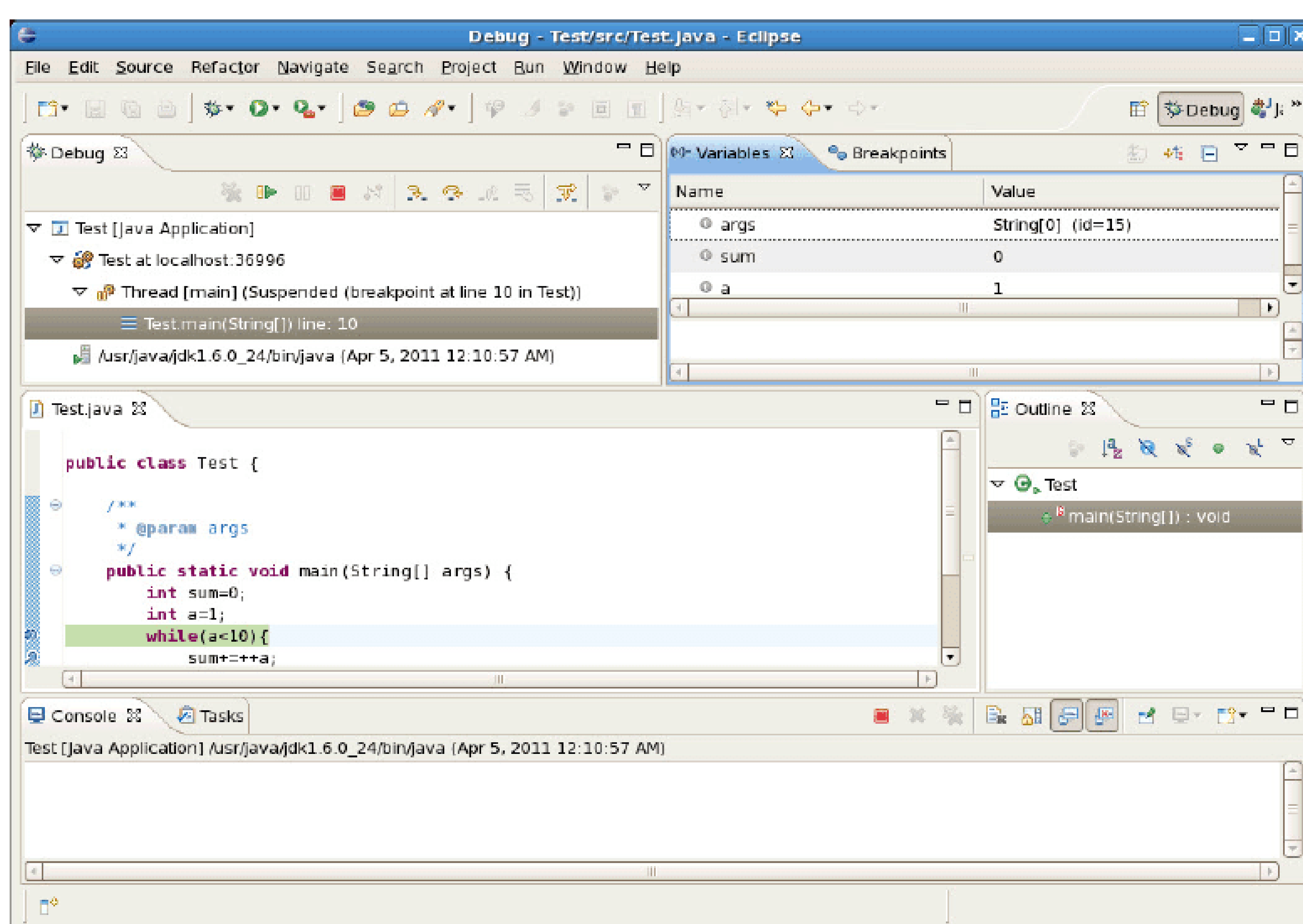


图 13.16　Debug 视图

在本例中，第一次循环执行完成后，通过 Variables 窗口中的变量可以看出问题。如图 13.17 所示。

在图 13.17 中两个变量的值都为 2，正确情况下，sun 值应该为 1，a 的值应该为 2，与编程过程中预期的值不符。查找代码中的错误，可以发现代码中错将 a++写为++a，因而导致了上述结果。排除上述问题后，继续执行可以发现循环条件有误，原本应为 a<=10 的条件语句被误写为 a<10，修改后即可完成代码的调试。

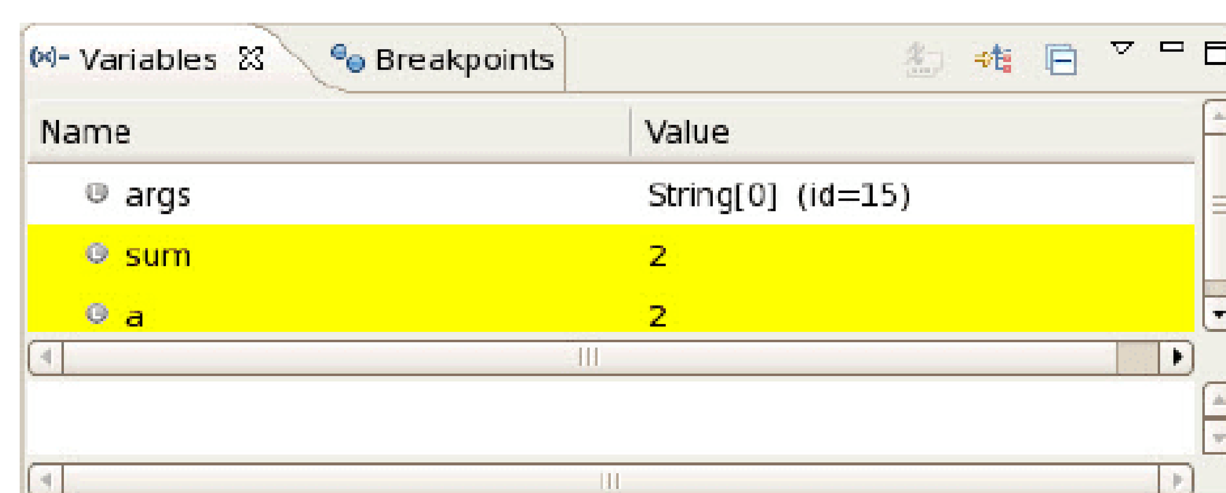


图 13.17　Variables 窗口

本小节中使用一个实例简单讲解如何使用 Eclipse 调试代码。在实际编写和调试过程中，可能遇到的问题和使用的断点十分多。在这种情况下，通常都推荐从小的功能模块入手逐一排除调试，以减小工作难度。

13.4　小　　结

- 13.1 节主要介绍了 Eclipse 开发平台历史概况，以及如何在 Linux 系统中安装 JDK、Eclipse 开发环境。
- 13.2 节简单介绍了 Eclipse 工作台的结构、Eclipse 中的窗口操作等。
- 13.3 节使用了一个简单的示例介绍了如何使用 Eclipse 开发程序、在 Eclipse 中如何提高输入代码的效率、运行和调试代码等内容。

Eclipse 是目前 Linux 系统中最成熟、应用最为广泛的开发环境，本章仅仅简单介绍了其基本功能和使用方法，感兴趣的读者可以参阅相关资料进一步学习。

第 14 章　常用的文本编辑器

在 Linux 系统中，除了前面介绍的 Vi、Vim、Emacs 等文本编辑器外，还存在一些很有特色的文本编辑器。这些编辑器可能不如 Vi、Vim 和 Emacs 功能强大且操作多样，但这些编辑器在某些方面却也有非常独到之处。

本章将介绍 Linux 系统中几个比较常见的文本编辑器，这些文本编辑器在各方面都具备自己的优势，涉及的主要内容如下。

- Nano 编辑器及其安装、使用方法等。
- Gnome 桌面环境中自带的 Gedit 文本编辑器的使用方法。
- KDE 桌面环境中自带的 Kate 编辑器使用方法及 Kate 中的脚本编程环境。

14.1　Nano 编辑器

Nano 是一个类 UNIX 系统中使用的文本编辑器。与其他文本编辑器相比，Nano 没有华丽的界面，也没有众多复杂的模式和快捷键。但 Nano 却拥有最简便的操作，编辑简单文本时，效率非常高，每一个初学者都可以很快上手。本节将简单介绍 Linux 系统中最简便的文本编辑器 Nano。

14.1.1　Nano 编辑器简介

Nano 编辑器是从 Pico 中发展起来的。Pico 是 UNIX 系统中的文本编辑器，由华盛顿大学开发。虽然 Pico 是一个免费软件，但却不是开源软件。

Nano 于 1999 年首次发布，但发布时其名称并不是现在的 Nano，而是 TIP（TIP isn't Pico），直到 2000 年才更名为 Nano。现在 Nano 属于 GNU 工程的一部分，并使用 GPL 作为其许可证。关于 Nano 的更多内容，感兴趣的读者可以查看其官方网站：http://www.nano-editor.org/。

由于 Nano 非常容易上手（仅需要数分钟即可上手），目前 Gentoo（一个允许用户自行选择软件的 Linux 发行版）和 Debian 都使用 Nano 作为其默认的文本编辑器。

14.1.2　安装 Nano 编辑器

默认情况下，RHEL5.3 中并没有安装 Nano 编辑器，因此必须要进行手动安装。本小节将简单介绍如何安装 Nano 编辑器。

（1）下载 Nano 安装包

在安装之前需要获得 Nano 编辑器的安装包：

```
#使用 wget 工具下载 Nano 编辑器安装包
# wget http://www.nano-editor.org/dist/v2.2/RPMS/nano-2.2.6-1.i386.rpm
```

上面的命令使用 wget 工具将安装文件 nano-2.2.6-1.i386.rpm 下载到当前目录中。

（2）安装 Nano 编辑器

下载完成之后可以使用以下命令进行安装：

```
#使用 rpm 命令安装 Nano 编辑器
# rpm -ivh nano-2.2.6-1.i386.rpm
Preparing...            ########################################### [100%]
   1:nano               ########################################### [100%]
```

完成上述步骤之后，Nano 编辑器就已经安装到系统中了。

14.1.3　Nano 启动及工作界面

使用编辑器之前，还应该了解如何启动 Nano 及 Nano 工作界面。

1．启动Nano编辑器

与 Vim 和 Emacs 等字符界面中的文本编辑器一样，Nano 也可以通过输入命令启动：

```
nano [option] filename
```

在使用此命令时，Nano 编辑器将会打开指定的文本文件。如果不指定要打开的文件名，Nano 将会打开一个未命名的新文本。

除此之外，启动 Nano 编辑器时，还可以使用一些选项，读者可以查阅帮助手册了解这些选项，此处不再一一赘述。

2．Nano编辑器的工作界面

使用 nano 命令启动 Nano 编辑器，其工作界面如图 14.1 所示。

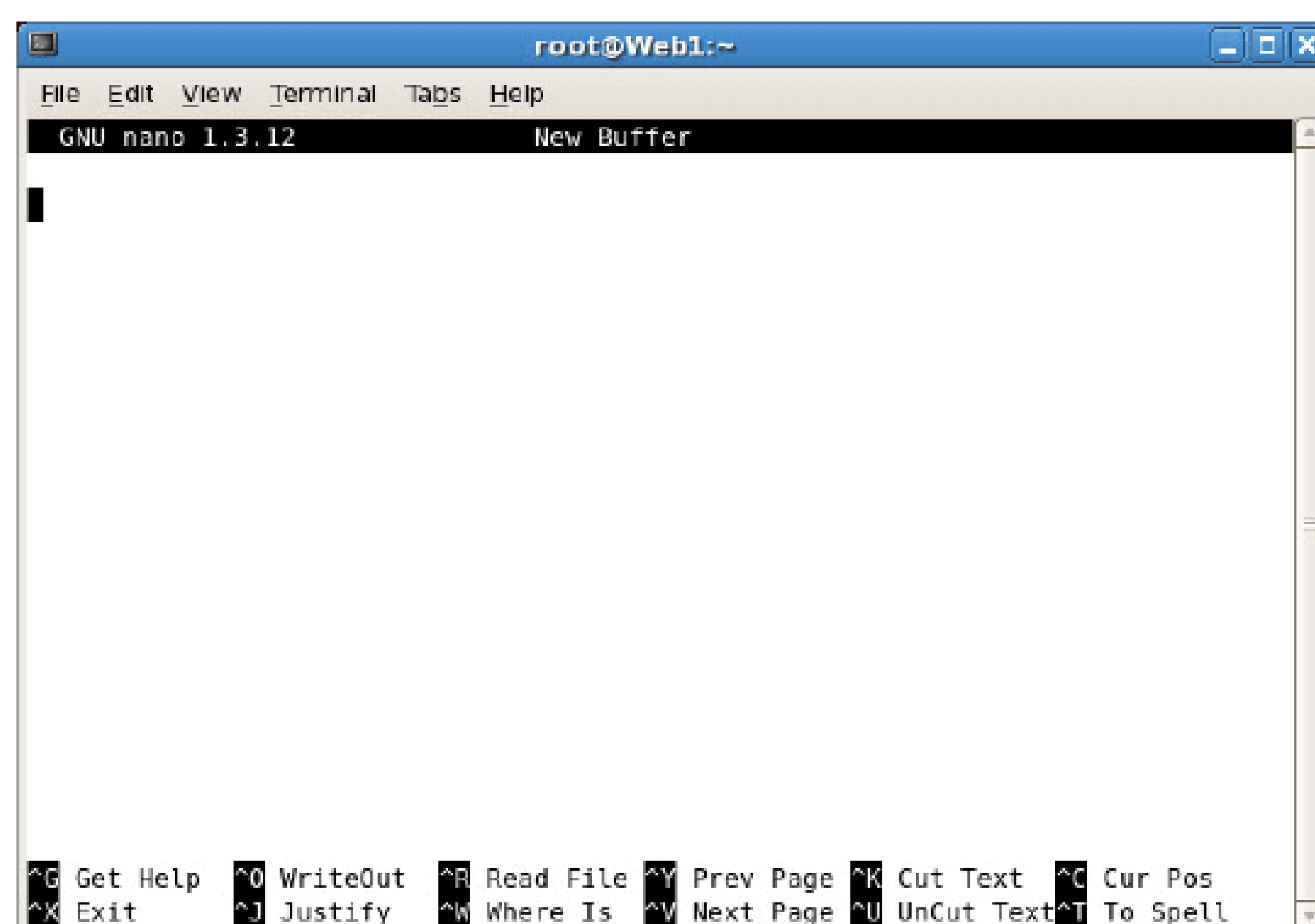


图 14.1　Nano 编辑器工作界面

从图 14.1 中可以看出，Nano 编辑器的工作界面非常简洁。文本编辑器的第 1 行用于显示编辑器的状态，包括 Nano 编辑器的版本、文件等信息。其中 New Buffer 表示当前打

开的是一个未命名的空白文本。在编辑器的第 1 行下面是供用户编辑文本内容的编辑区，用户可以在此阅读文本的内容，也可以在此编辑文本。编辑器的最后两行是快捷键显示栏，其中显示的是用户使用频率最高的几个快捷键。

提示：快捷显示栏中的^表示键盘上的 Ctrl 键，例如^G 表示快捷键 Ctrl+G。

3．使用帮助

在 Nano 编辑器中按下 Ctrl+G 或 F1 键可以打开编辑器帮助界面，如图 14.2 所示。

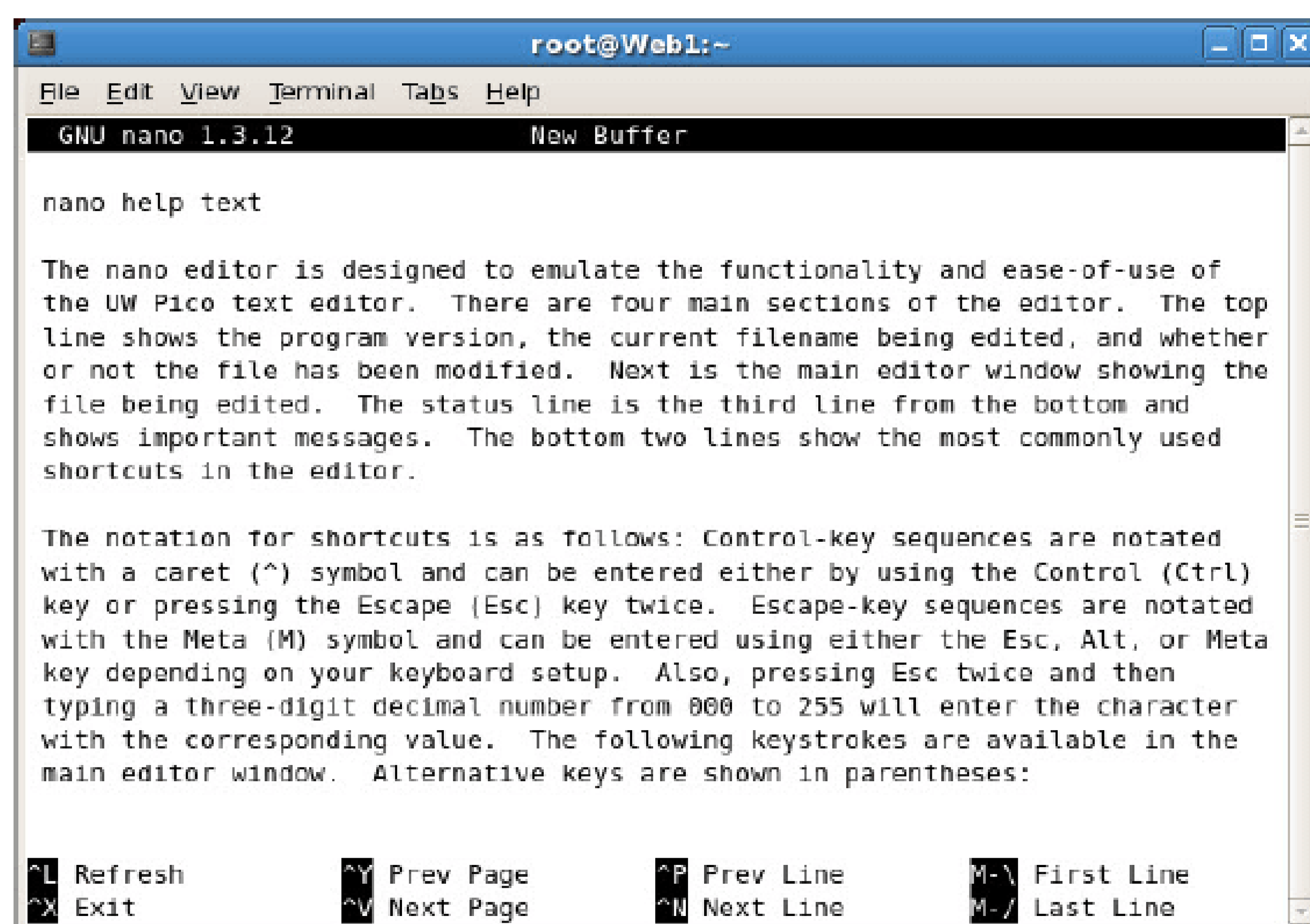


图 14.2　Nano 的帮助界面

在 Nano 帮助界面中用户可以查看编辑器的基本信息，最下方还显示了在帮助界面中可以使用的快捷键。

- ❑ Ctrl+Y、Ctrl+V：向上翻页、向下翻页。
- ❑ Ctrl+P、Ctrl+N：向上翻动一行、向下翻一行。
- ❑ Ctrl+X：退出当前帮助界面。

Nano 编辑器的帮助信息非常简明，读者可以自行阅读，了解 Nano 编辑器的相关信息。

4．退出Nano编辑器

退出 Nano 编辑器可以使用快捷键 Ctrl+X，此时如果编辑器中的内容有所改变，Nano 将会提示用户是否保存，如图 14.3 所示。

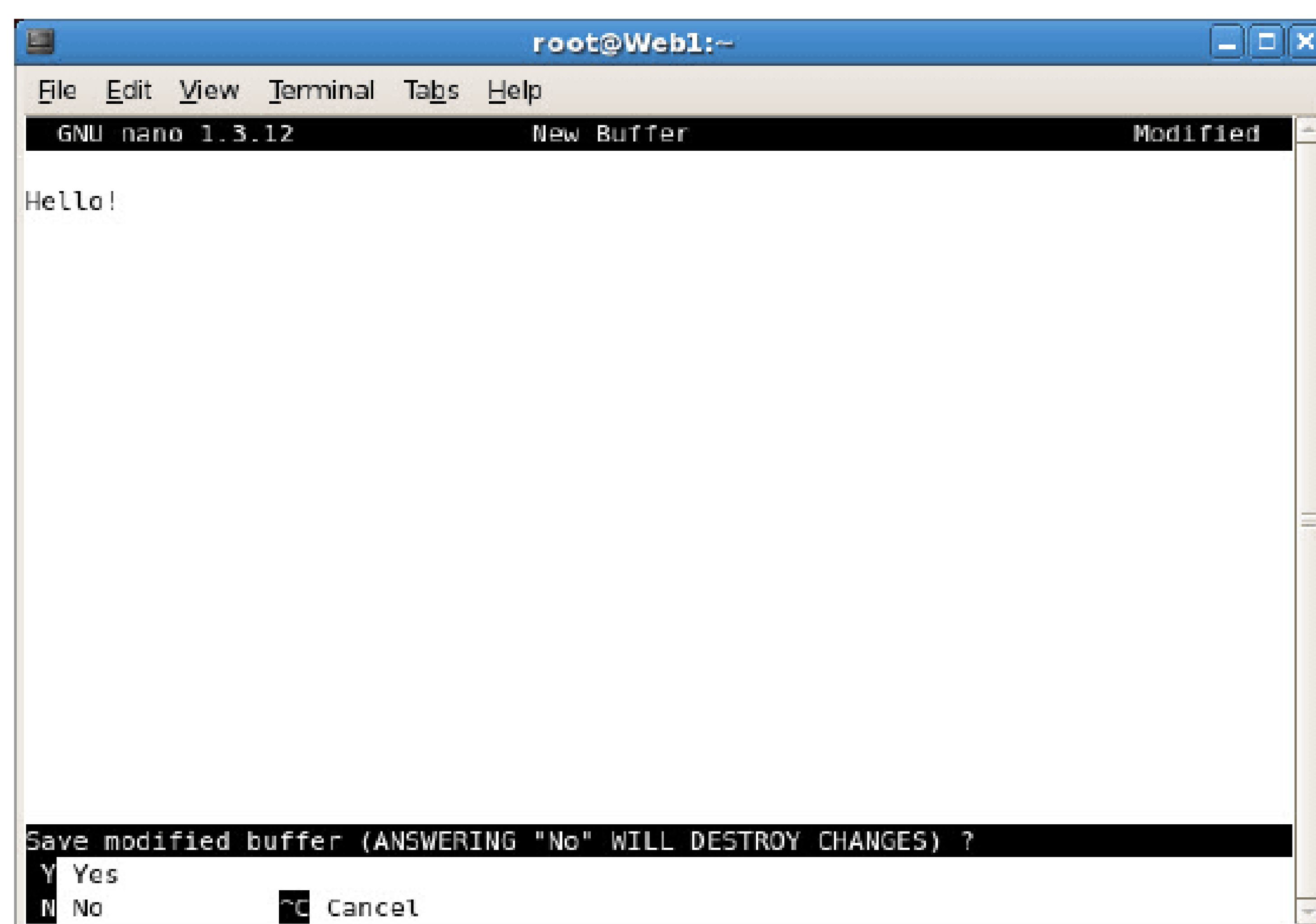


图 14.3　退出时提示是否保存文本

此时用户可以执行的操作如下。

- 输入字母 Y 并按下 Enter 键保存已改变的内容。如果文本尚未命名，Nano 会提示用户输入文本的名称及路径，输入完成后即可退出 Nano 编辑器。
- 输入字母 N 立即退出编辑器。此操作会丢失所有已经编辑的内容，在执行前应该确认内容已经没有任何意义。
- 按下快捷键 Ctrl+C 取消退出操作。此操作将会中止退出操作，用户可以在中止后继续编辑文本。

14.1.4　快速移动光标

在 Nano 编辑器中，许多操作都是通过快捷键实现的，例如保存、退出等。因此需要了解一些最基本的快捷键才能对文本进行编辑。从本小节开始将简单介绍 Nano 编辑器中的快捷键。

许多时候需要快速移动文本编辑器中的光标，以便将光标移动到需要编辑的位置。在 Nano 编辑器中，可以使用方向键在编辑器中快速移动光标。除此之外，还可以使用以下快捷键快速移动光标。

- Ctrl+Y、Ctrl+V：向上、向下翻页，也可以使用编辑键 Page Up、Page Down 替代。
- Ctrl+P、Ctrl+N：向上、向下翻动一行。
- Ctrl+A、Ctrl+E：快速移动光标至行首、行尾，也可以使用编辑键 Home、End 替换。
- Alt+\、Alt+/：快速移动光标至文本首、文本尾。
- Ctrl+F、Ctrl+B：快速向后、向前移动光标。

除此之外，Nano 中还有一些不常用的移动光标快捷键，此处不再一一介绍，感兴趣的读者可以阅读相关文档。

14.1.5　复制粘贴文本

复制和粘贴文本是文本编辑器中最基本的功能之一，虽然 Nano 编辑器工作在字符界面中，但其仍然提供了非常灵活而又方便的复制、粘贴功能。在 Nano 编辑器中，除了提供复制、剪切整行的行复制、剪切功能之外，还提供了可以复制、剪切任意文本的自由复制、剪切功能。本小节将简单介绍 Nano 编辑器中的这些复制、剪切功能。

1．行复制、剪切

在 Nano 编辑器中，复制、剪切的快捷键如下。

- Alt+6：将当前光标所在行复制到缓冲区中。
- Ctrl+K：将光标所在行剪切到缓冲区内，这个操作将会立即删除当前行。
- Ctrl+U：将缓冲区中的内容粘贴到光标所在位置。

这两个快捷操作与 Vim 中的复制粘贴类似，此处不再赘述其用法。

2．自由复制、剪切

Nano 编辑器还有如下几个更加灵活的复制、剪切快捷键。

- ❑ Ctrl+6：设置复制文本的起始位置。
- ❑ Alt+A：设置剪切文本的起始位置。
- ❑ Alt+6：设置复制文本的结束位置。
- ❑ Ctrl+K：设置剪切文本的结束位置。
- ❑ Ctrl+U：粘贴文本。

自由复制、剪切功能的使用方法是将光标移动到需要复制、剪切的起始位置，然后按下 Ctrl+6、Alt+A 快捷键标记复制、剪切的起始位置。然后移动光标，此时 Nano 会将当前光标与起始位置之间的文本以黑底白字的形式显示，如图 14.4 所示。

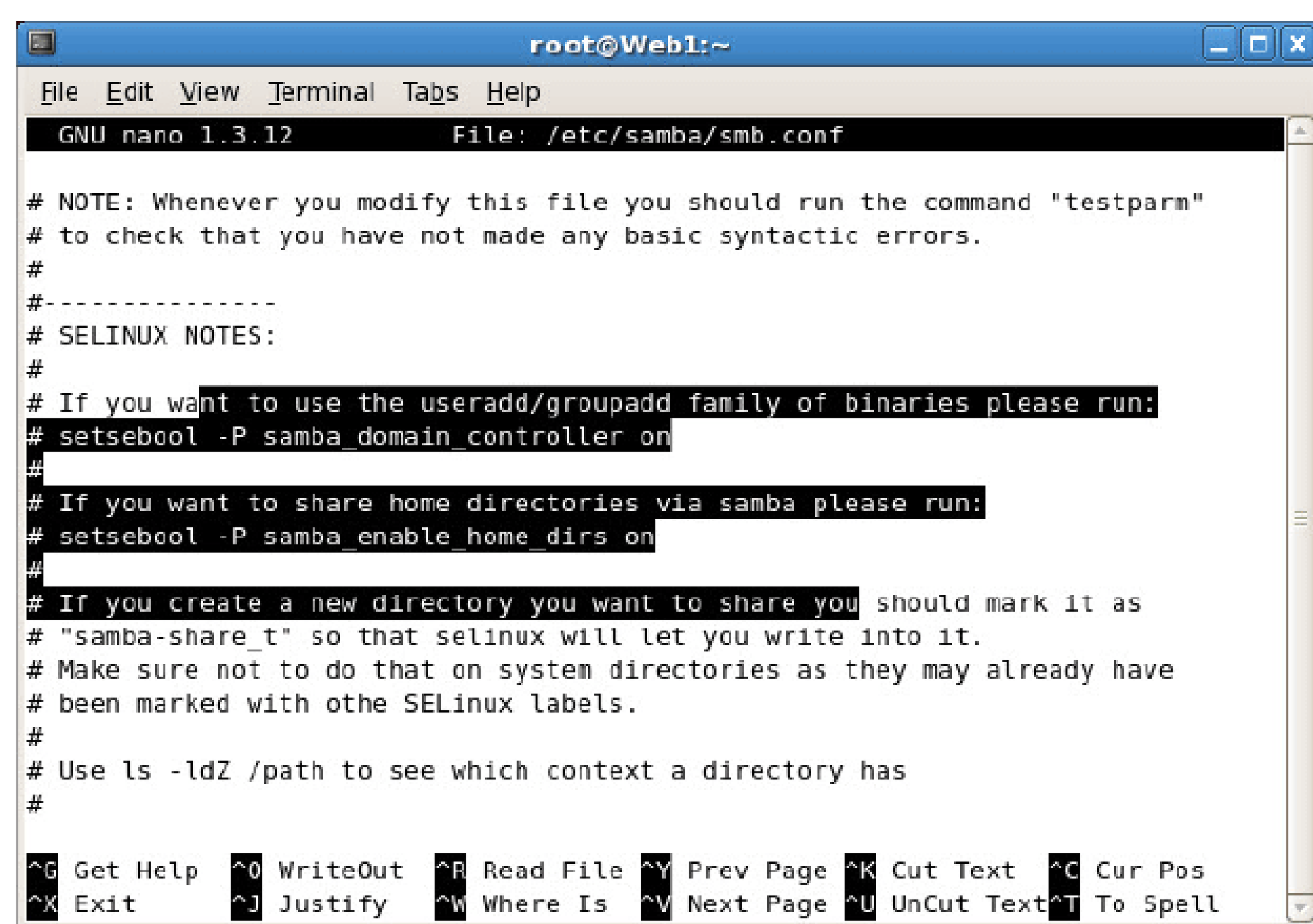


图 14.4　突出显示的复制内容

当光标移动到需要复制、剪切的文本结尾时，按下 Alt+6、Ctrl+K 快捷键对所选内容进行复制、剪切。然后将光标移动到目标位置，使用 Ctrl+U 快捷键进行粘贴即可。

14.1.6　查找和替换

为了方便用户对文本进行编辑，现在的文本编辑器都附带了查找和替换功能，使用这两个功能可以更快地定位和修改文本。本小节将简单介绍如何使用 Nano 编辑器的查找和替换功能。

1．查找文本

在 Nano 编辑器中，查找文本的快捷键是 Ctrl+W。按下查找快捷键后，Nano 将会提示用户输入要查找的文本，如图 14.5 所示。

此时可以输入要查找的文本并按下 Enter 键，Nano 将向下查找文本并将光标停留在找到的第 1 个文本处。如果需要跳转到下一个查找到的文本，可以先按下 Ctrl+W 键后再按 Enter 键，Nano 会将光标停留在下一个找到的文本处。

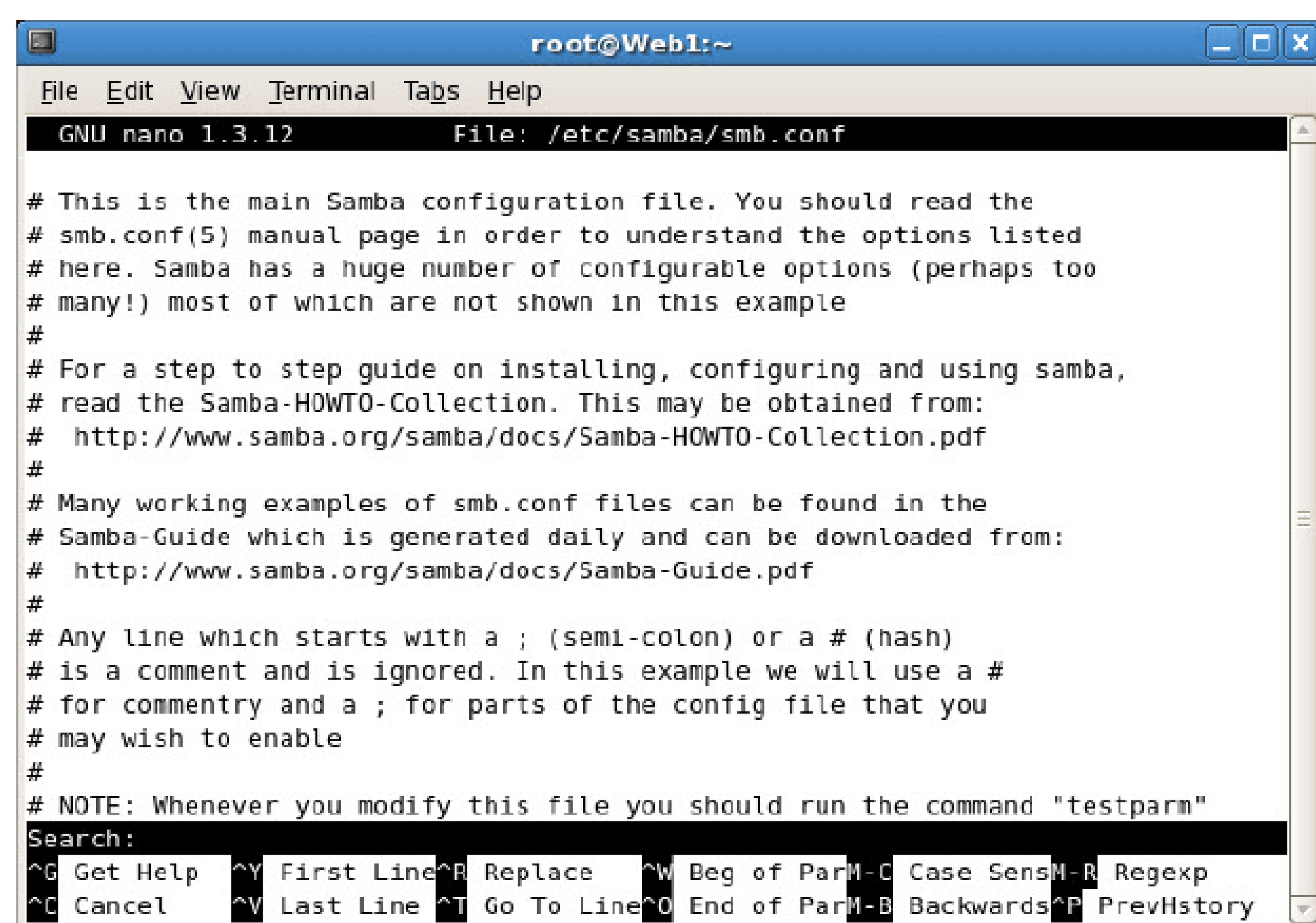


图 14.5　查找文本

提示：按下查找快捷键时，如果此前使用过查找功能，Nano 编辑器会将上一次查找的文本设置为默认要查找的文本（按下 Enter 键即可使用默认查找的文本）。

2. 查找并替换文本

查找并替换文本使用的快捷键是 Ctrl+\，其使用方法与查找文本功能类似，先按下快捷键 Ctrl+\，Nano 编辑器将提示用户输入要查找的文本，如图 14.6 所示。

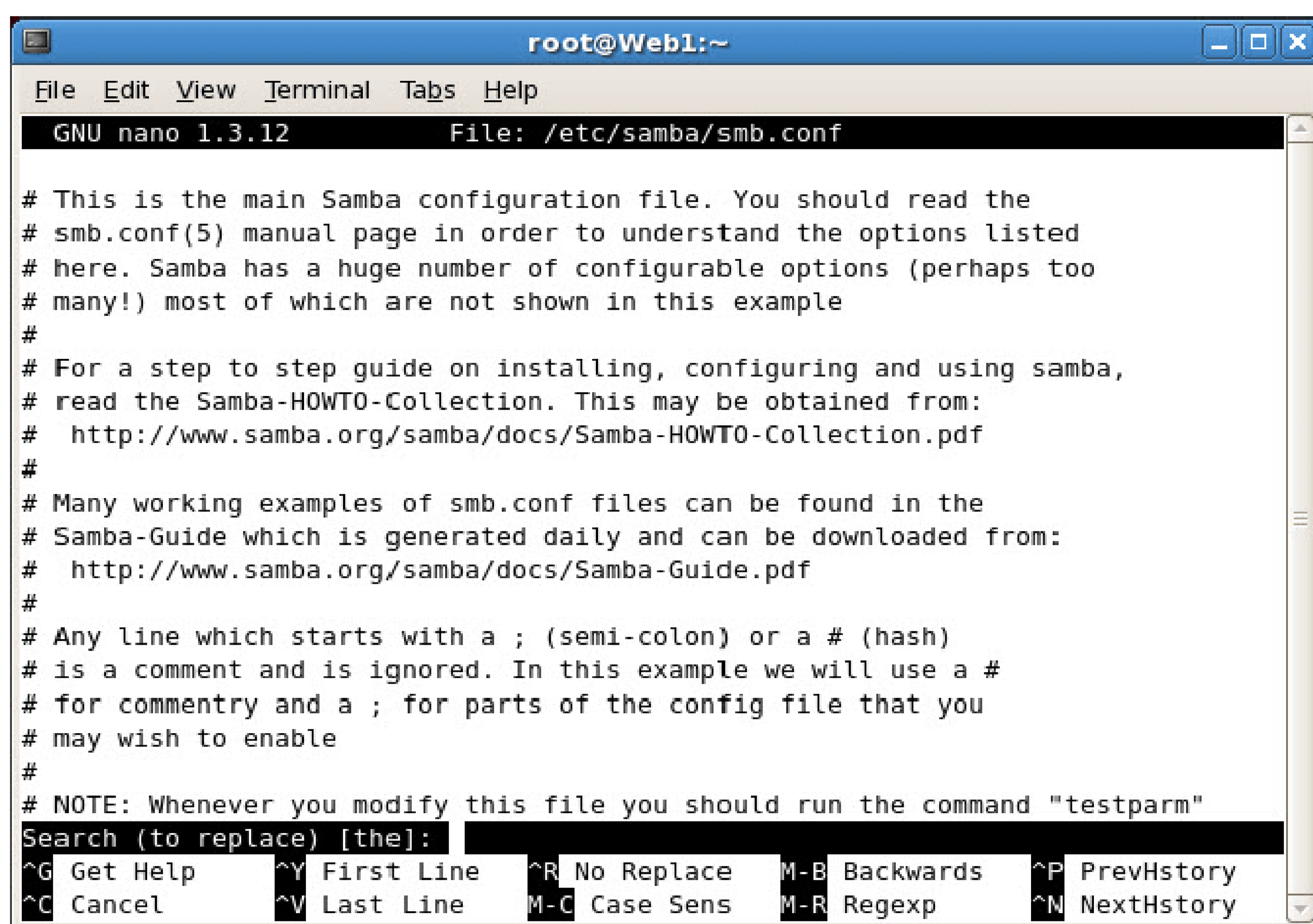


图 14.6　查找并替换文本

在图 14.6 中，Nano 编辑器使用了上一次查找的文本 the 作为默认查找文本，如果此时用户按下 Enter 键，Nano 编辑器将会使用这个默认值作为查找的文本。输入要查找的文本之后按下 Enter 键，然后输入替换后的文本，并按下 Enter 键之后，Nano 编辑器就开始查找文本并进行替换了。

为慎重起见，Nano 编辑器查找到文本后并不会直接替换，而会提示用户对当前查找到的文本如何处理，如图 14.7 所示。

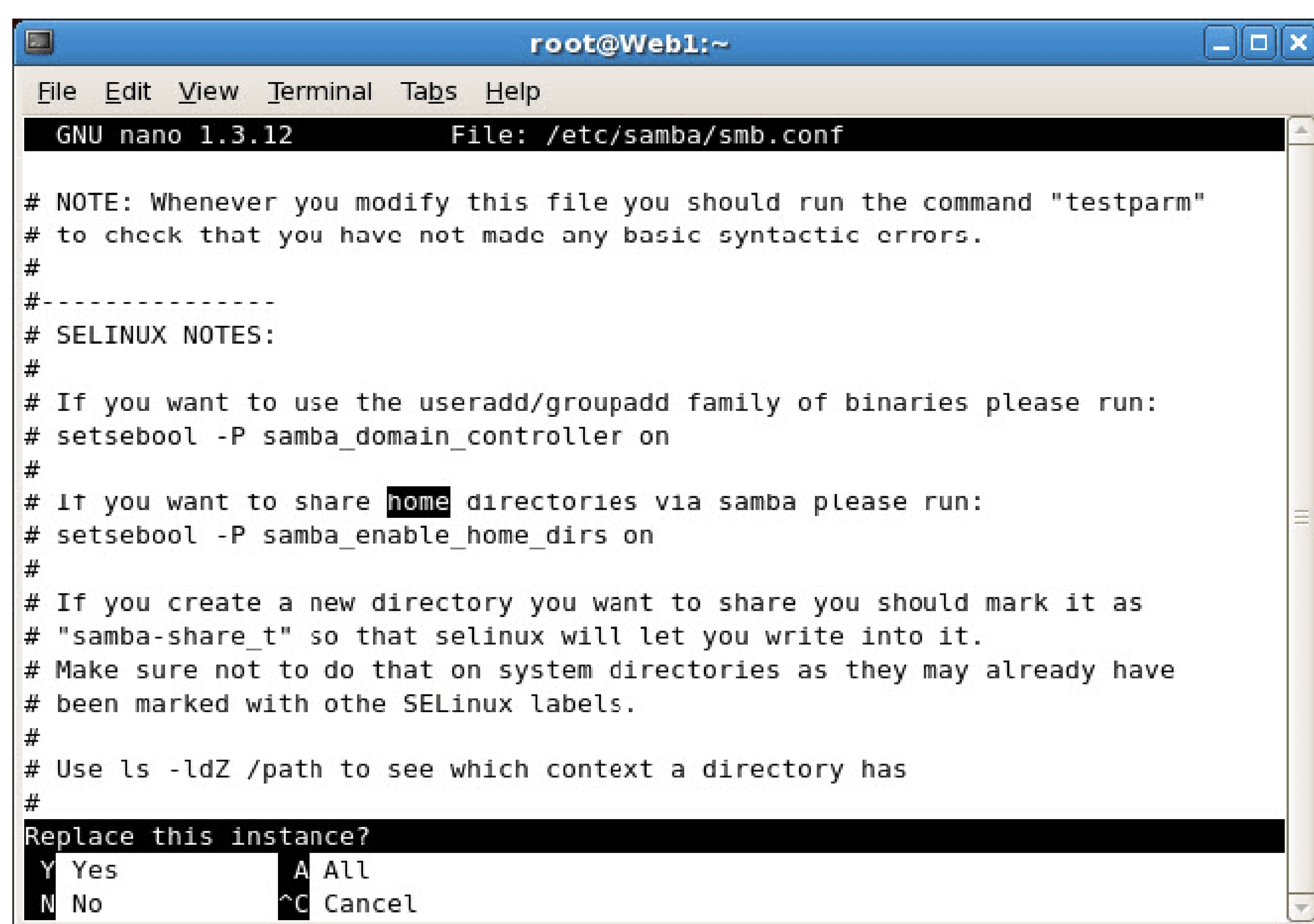


图 14.7　查找并询问用户

此时用户可以进行如下操作。

- 按下 Y 键替换当前文本。
- 按下 N 键跳过当前查找到的文本，继续查找下一个。
- 按下 A 键对所有查找到的文本进行替换。如果所有的文本都需要进行替换，可以使用 A 键，使用此键前应该确认所有情况都已被排除。
- 使用快捷键 Ctrl+C 取消。如果文本中所有需要替换的文本都已经替换完成，可以使用此快捷键退出查找替换模式。

提示： 本节仅简单介绍 Nano 编辑器的基本功能，除此之外，还有许多其他的功能（例如拼写检查等）。这对于一个安装包仅有 590KB 的软件而言，非常不寻常。感兴趣的读者可以阅读相关文档进一步学习 Nano 编辑器。

14.2　Gedit 文本编辑器

使用 Gnome 桌面环境的 Linux 发行版中，通常都装有一个图形化的文本编辑器 Gedit。虽然 Gedit 编辑器被默认安装在系统中，但大多数人都仅用到了其基本功能，Gedit 除了最简单的文本编辑外，还可以自动备份文本、用来进行各种语言编程等。本节将介绍 Gnome 桌面环境中的文本编辑器 Gedit 的使用方法。

14.2.1　Gedit 文本编辑器概述

准确地说 Gedit 并不是一个独立的 GNU 项目，它是 GNU 桌面环境 Gnome 项目的一部分，因此 Gedit 通常都安装在 Gnome 桌面环境中。Gnome 是 Linux 系统中最常见的桌面系统之一，RHEL5.3 中默认的桌面环境正是 Gnome。Gnome 桌面环境和 Gedit 文本编辑器的图标如图 14.8、图 14.9 所示。



图 14.8　Gnome 图标



图 14.9　Gedit 图标

对于用户而言，Gedit 文本编辑器有许多好处。

- ❑ 兼容 UTF-8 等多种编码模式，能够很好地支持多种语言。
- ❑ 可以设置多种编程语言语法高亮。
- ❑ 支持打印和打印预览功能。
- ❑ 支持文本自动缩进功能。

关于 Gedit 文本编辑器的更多内容此处不再一一介绍，感兴趣的读者可以查阅其官方网站了解：http://projects.gnome.org/gedit/。

14.2.2　Gedit 工作界面介绍

如果安装的 RHEL5.3 使用的图形界面是 Gnome，那么 Gedit 文本编辑器已经被安装到系统中了。本小节将简单介绍 Gedit 文本编辑器的启动、退出及工作界面等内容。

1．启动Gedit文本编辑器

在 Gnome 桌面环境中，依次单击桌面左上角的 Applications→Accessories→Text Editor，即可启动 Gedit 文本编辑器，如图 14.10 所示。

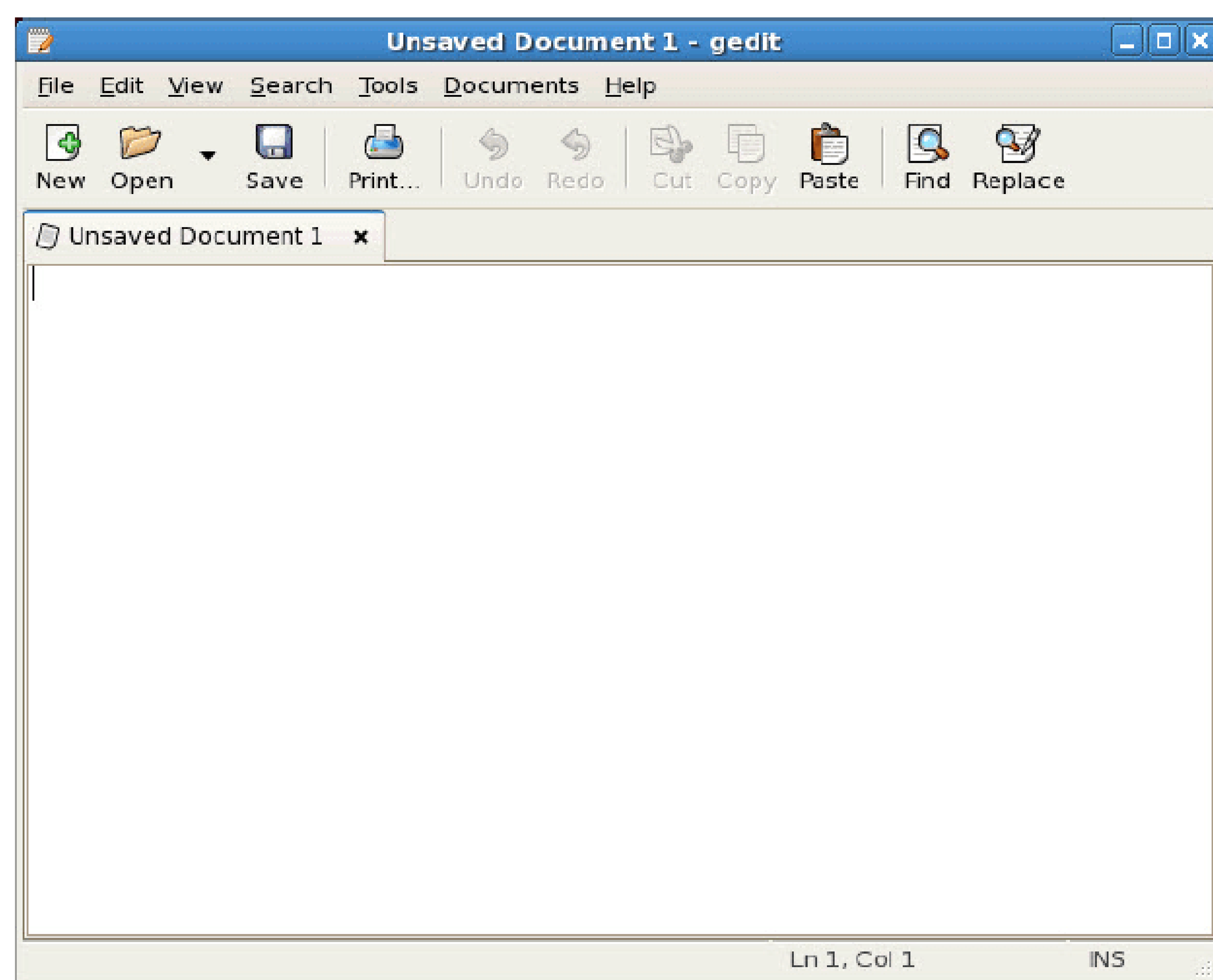


图 14.10　Gedit 文本编辑器

2．Gedit编辑器工作界面

Gedit 文本编辑器的工作界面与 Windows 系统中的文本编辑软件类似，从上至下依次是标题栏、菜单栏、工具栏、标签栏、工作区及状态栏。与 Windows 系统中的文本编辑器记事本不同，Gedit 可以在一个窗口中通过切换标签的方式同时编辑多个文本。

14.2.3　快速移动光标

在 Gedit 编辑器中，可以像 Windows 中的文本编辑器那样，使用鼠标点击来定位需要编辑的文本位置，也可以使用右侧滚动条快速翻页。本小节将简单介绍 Gedit 编辑器中移动光标的其他方法。

（1）快速移动光标

在 Gedit 编辑器中，除了使用方向键、鼠标移动光标之外，还可以使用以下编辑键快速移动光标。

- Home、End 键：将光标移动到行首、行尾。
- Page Up、Page Down 键：向前、向后翻页。

（2）快速跳转行

如果用户需要调试程序的源代码，那么快速跳转到行的定位方式将十分有用（如果程序出错，编译器通常会以行号的方式提示用户）。Gedit 也可以使用跳转行的方式定位文本，要跳转到指定的行，可以依次单击菜单栏中的 Search→Go to Line。此时 Gedit 编辑器将会弹出一个窗口，要求用户输入要跳转的行号，如图 14.11 所示。

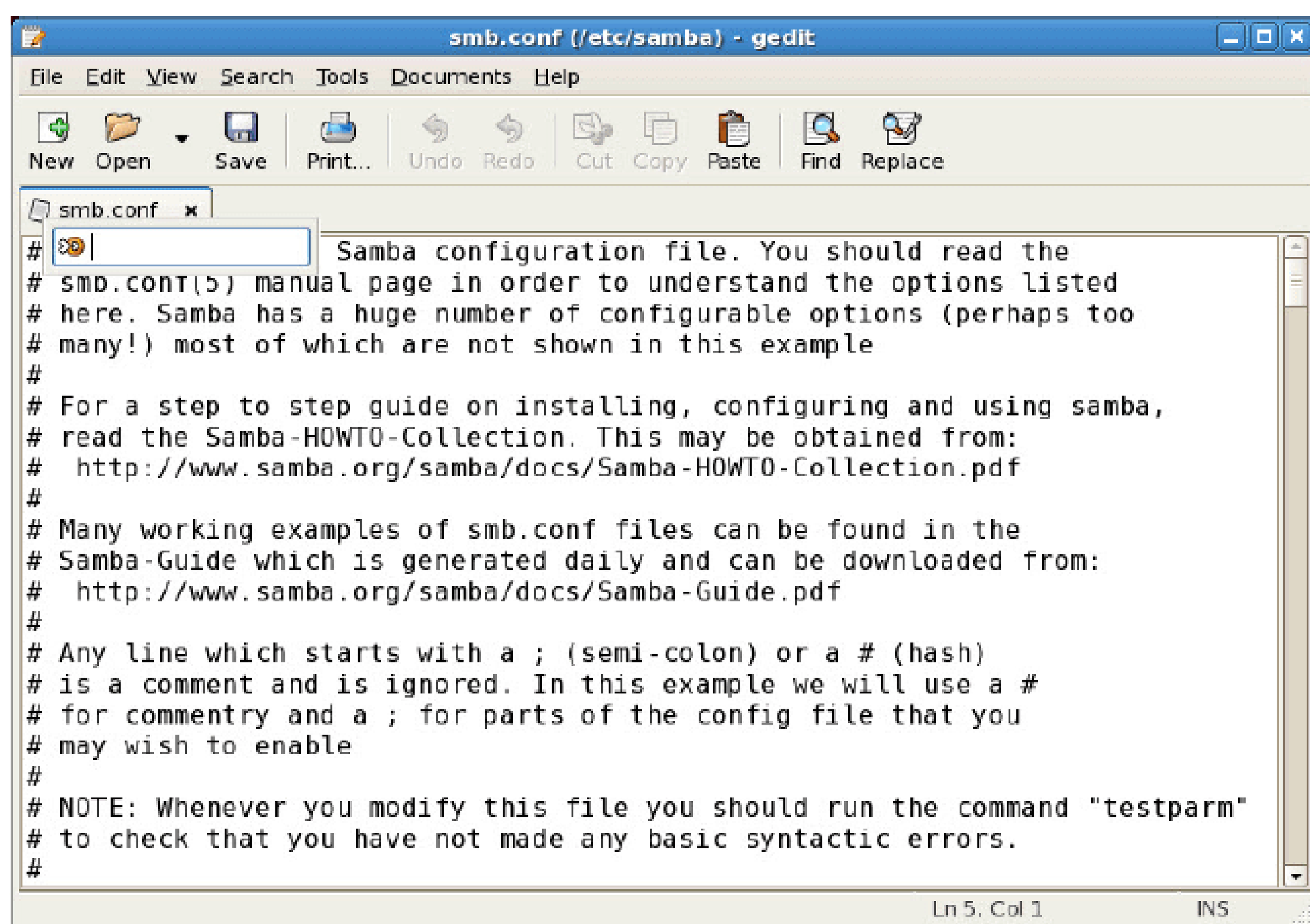


图 14.11　跳转到行

此时输入要跳转的行号即可。

14.2.4　显示行号

在编写程序代码时，可能用户更希望在编写或查看时能够显示这些代码的行号，以便按提示查看或发现代码中的错误。要在 Gedit 编辑器中显示行号，可以依次单击菜单栏中的 Edit→Preferences，此时将会弹出 gedit Preferences 对话框，如图 14.12 所示。

此时可以点选 View 标签中的 Display line numbers，并单击 Close 按钮，此时 Gedit 编辑器将会显示行号。

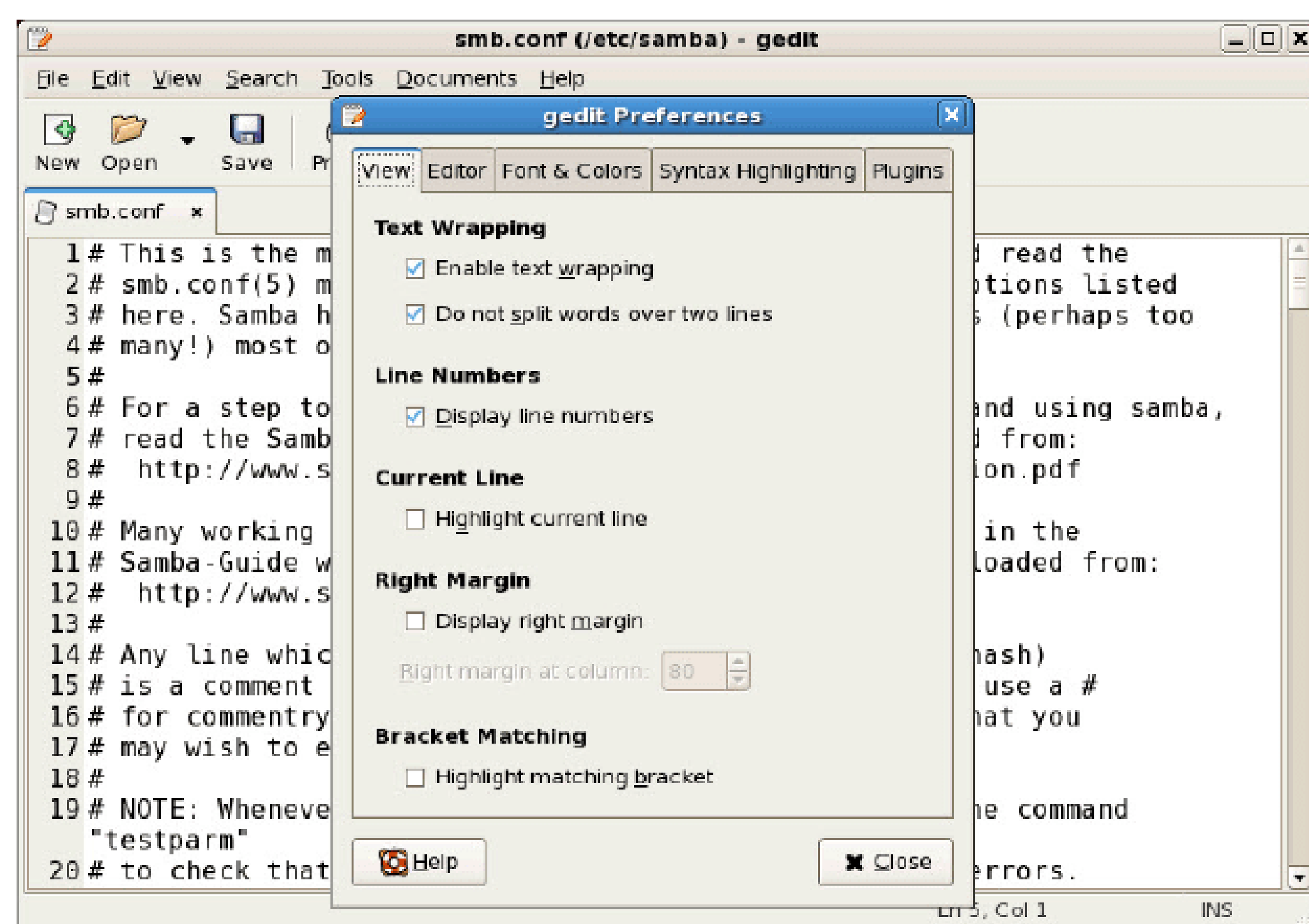


图 14.12　gedit Preferences 对话框

14.2.5　语法高亮

Gedit 支持多种编程语言的语法检查，文本编辑器语法检查是指以高亮的形式显示编程语言中的关键字（称为语法高亮）。用户可以通过语法检查的结果查看代码中存在的错误。

要在 Gedit 编辑器中使用语法高亮，可以依次单击菜单栏中的 View→Highlight Mode→Source，此时会列出支持语法高亮的编程语言。按需要选择编程语言即可，如图 14.13 所示。

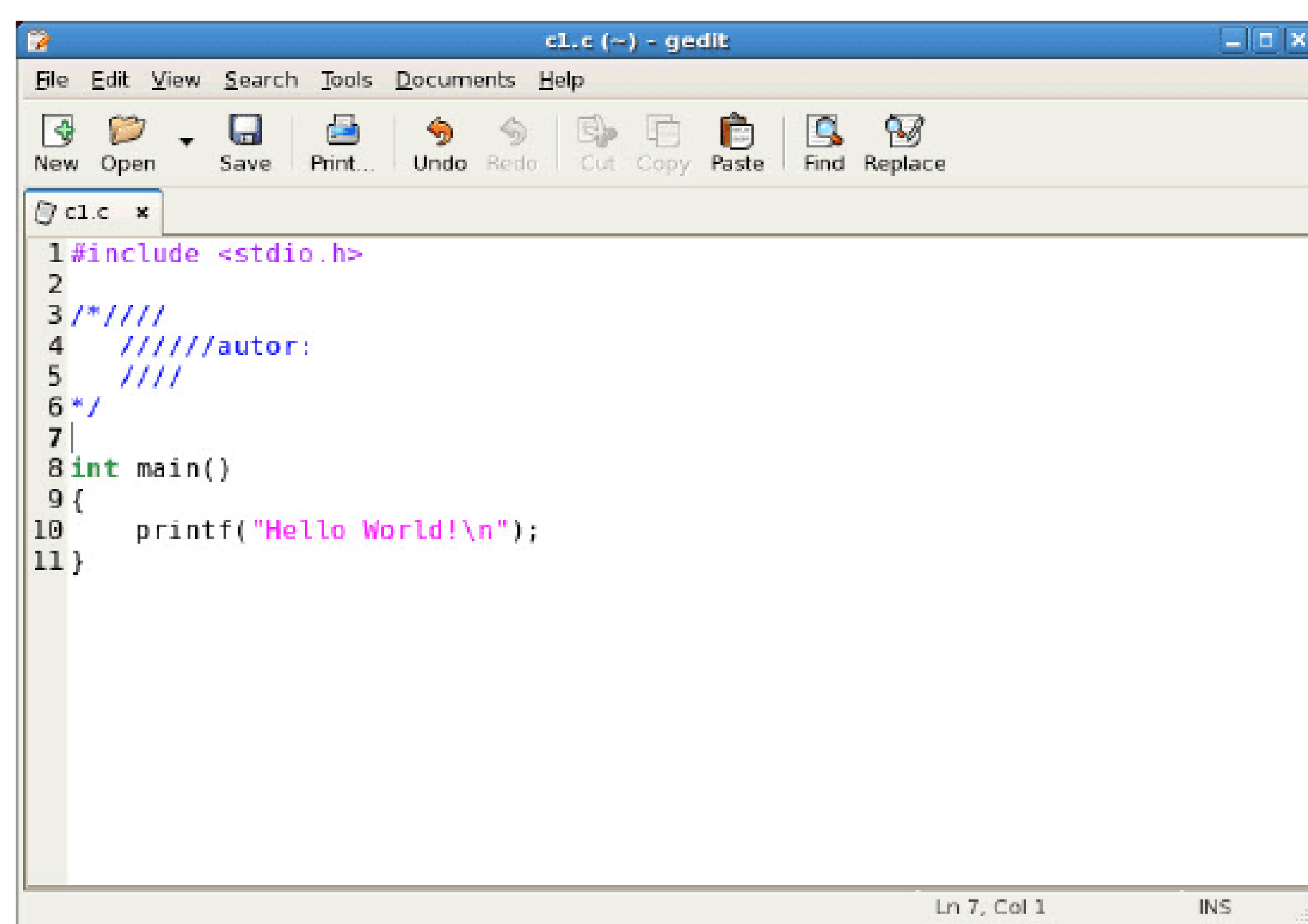


图 14.13　语法高亮

如果需要关闭语法高亮功能，依次单击菜单栏中的 View→Highlight Mode→None 即可。

14.2.6　拼写检查

许多时候用户都希望有一个工具能提示文本录入时的错误，特别是当输入一个错误的单词时。这时可以使用 Gedit 编辑器提供的拼写检查来帮助我们发现文本中的错误单词。要打开拼写检查，可以单击菜单栏中的 Tools，并勾选弹出菜单中的 Autocheck Spelling（自

动拼写检查）。此时 Gedit 编辑器将会检查文本中的单词，并使用红色波浪线标示出错误的单词，如图 14.14 所示。

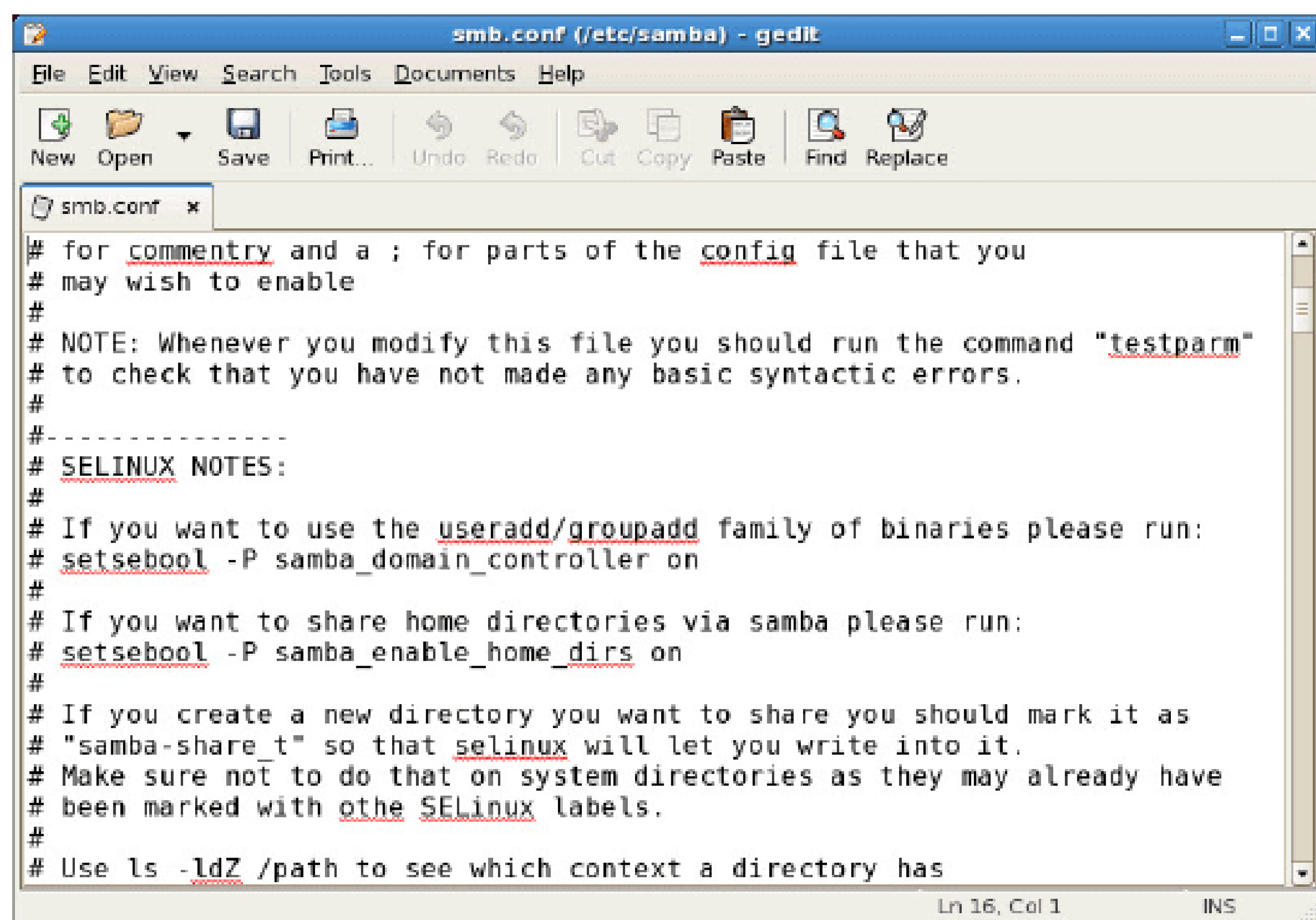


图 14.14　拼写检查

在图 14.14 中，Gedit 以红色波浪线的方式显示拼写有误的单词。需要注意的是，拼写检查主要依赖的是 Gedit 使用的词库，较新的单词、特殊用法也可能会被标记为错误。

有时虽然编辑器发现了单词拼写错误，但可能一时忘记正确的单词应该如何拼写，这时可以使用编辑器自带的拼写建议功能。在以红色波浪线标识的错误单词上单击鼠标右键，然后在弹出的快捷菜单中选择 Spelling Suggestions（拼写建议），编辑器将会显示当前错误单词的拼写修改建议，如图 14.15 所示。

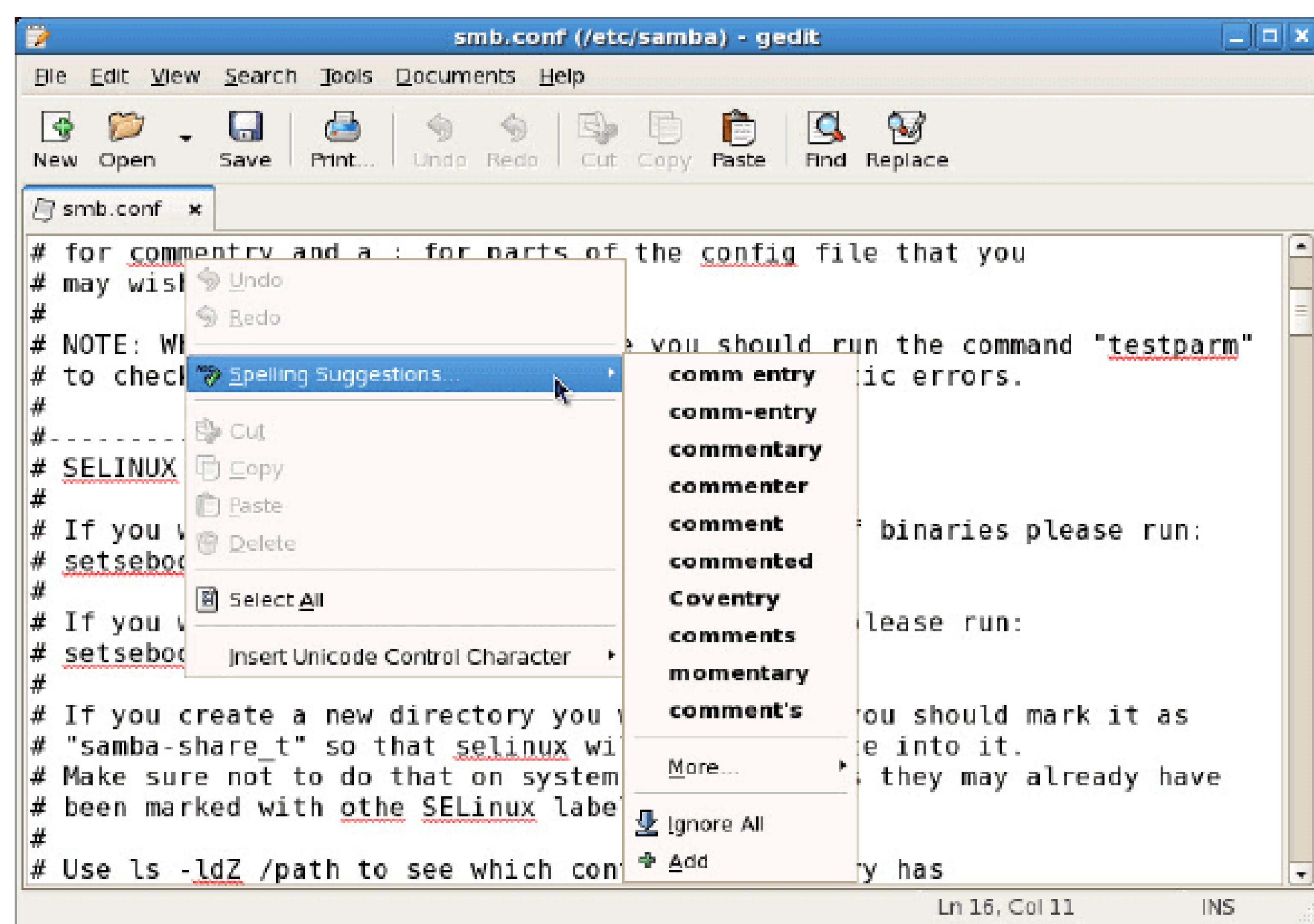


图 14.15　拼写修改建议

此时可以在拼写建议列表中选择正确的单词，如果单词不在列表中，可以单击 More 查看更多拼写建议。如果当前这个单词是一个特殊的单词，例如一些特殊专有名词，可以单击 Ignore All 忽略这个单词的拼写检查。当然最好单击 Add，将特殊专有名词添加到词库中，以后就不会再提示错误了。

注意： 在使用拼写检查时，可能会出现正确的单词也提示单词拼写错误的现象，这可能是当前编辑使用的语言有误。要更改编辑器使用的语言，依次单击 Tools→Set Language，在弹出的语言选择对话框中选择正确的语言即可。

14.2.7　查找和替换

对于用户而言，文本编辑器中的查找和替换功能十分重要，Gedit 编辑器也提供了此功能。本小节将简单介绍 Gedit 编辑器的查找和替换功能。

1. 查找文本

在 Gedit 编辑器中，查找分为非增量查找和增量查找两种。要使用非增量查找，可以单击菜单栏中的 Search→Find，在弹出的文本框中输入要查找的文本，即可进行非增量查找。如果查找到的文本不是需要查找的文本，可以使用 Search 菜单中的 Find Next 和 Find Previous 查找下一个和上一个文本。

要使用增量查找，可以依次单击菜单栏中的 Search→Interactive search，此时在弹出的对话框中输入要找的文本即可增量查找文本。

2. 替换文本

许多时候可能用户发现某些文本输入错误，需要批量修改这些文本，此时可以使用替换文本功能替换这些错误的文本。要使用替换文本功能，可以单击菜单栏中的 Search→Replace 或单击工具栏的替换按钮打开替换文本对话框，如图 14.16 所示。

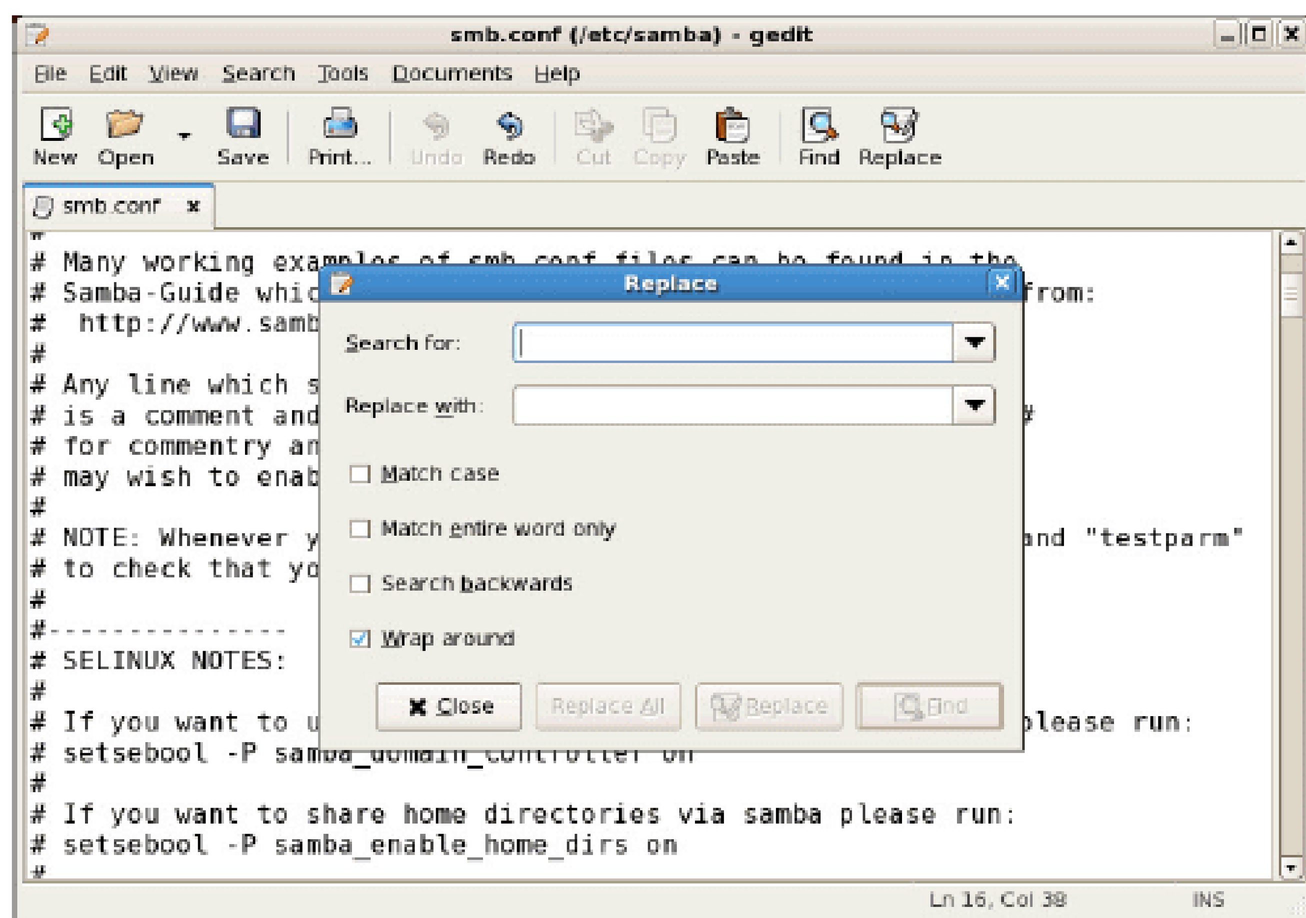


图 14.16　替换对话框

此时按需要输入要查找的文本和替换后的文本即可使用 Gedit 编辑器的替换功能。在替换时，还可以按需要选择 Match case（匹配大小写），Match entire word only（匹配整个单词），Search backwards（反向查找），Wrap around（越过文件头、尾查找），执行更精确的查找。

提示： 图形环境下的 Gedit 编辑器，与 Windows 系统中的文本编辑器记事本的操作方法十分类似，本书中不一一列举其使用方法，读者可以通过阅读相关文档了解更多的使用技巧。

14.3　Kate 编辑器

Kate 编辑器（KDE Advanced Text Editor，KDE 高级文本编辑器）是 KDE 桌面环境中的文本编辑器。与 Gedit 编辑器一样，Kate 也是 KDE 桌面环境中自带的文本编辑器。虽然 Kate 是桌面环境下自带的编辑器，但其功能十分强大。不仅集成了用于程序设计、配置文本、网页设计、数据库等特殊文本的语法高亮功能，还集成了文件查找功能、终端、文档管理器等工具。本节将简单介绍 KDE 桌面环境中的 Kate 编辑器。

14.3.1　Kate 编辑器概述

KDE（Kool Desktop Environment）是类 UNIX 系统中最为流行的桌面环境之一，现在许多 Linux 发行版都默认包含 KDE 桌面环境，其华丽的界面为其赢得了不少的用户。作为其组件之一的 Kate 文本编辑器，从 KDE 2.2 版本开始就已经成为其中的一员。目前 KDE 桌面环境发布的最新版本为 4.4，但在 RHEL5.3 中，系统为用户安装的版本号是 3.5，如图 14.17 所示。

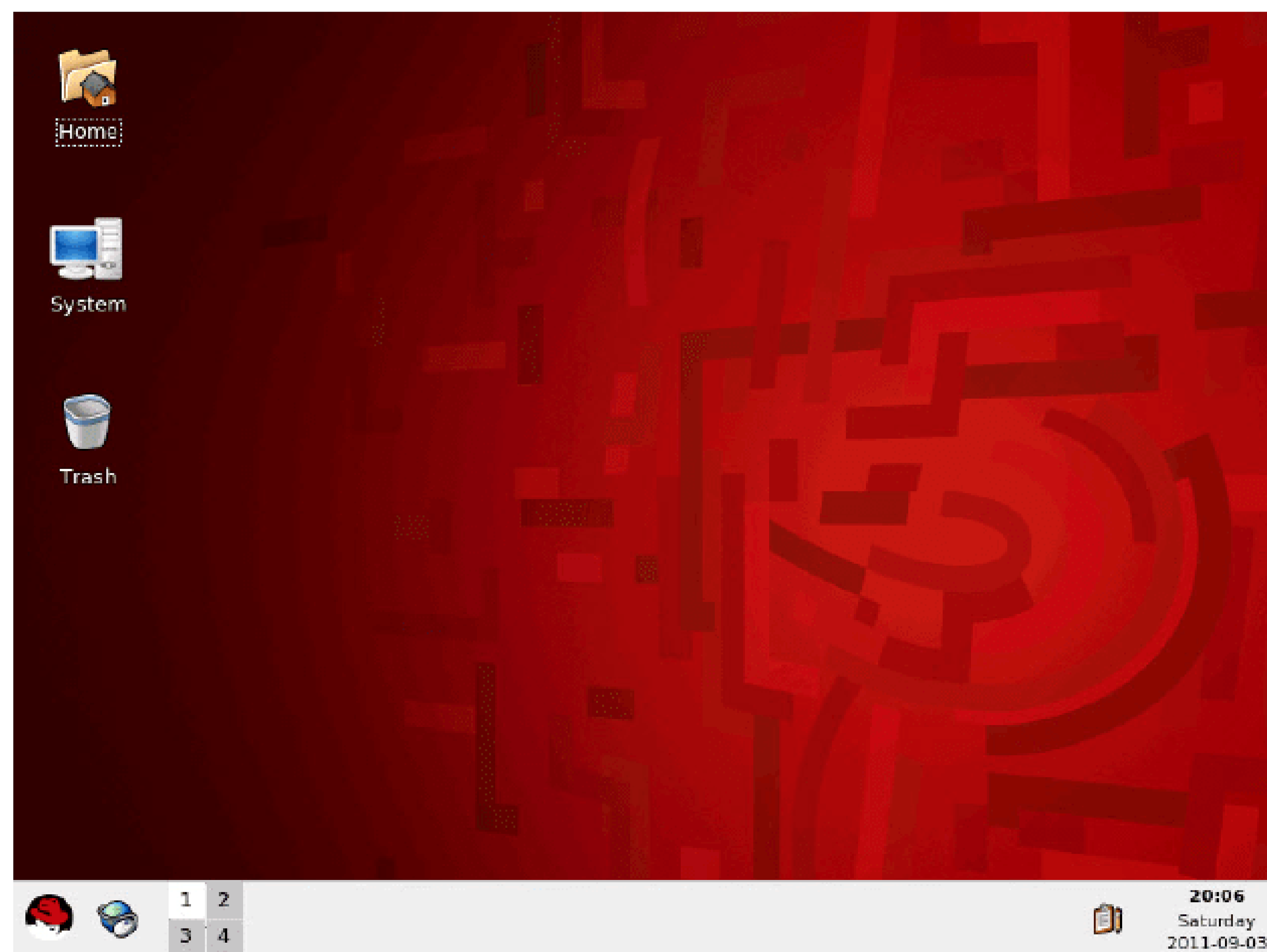


图 14.17　RHEL5.3 中的 KDE 桌面环境

提示：KDE 及其桌面环境图片相关内容来自 KDE 官方网站。

关于 KDE 桌面环境及 Kate 编辑器的更多内容，读者可以参考其官方网站。

- KDE 中文网址：http://www.kdecn.org/
- KDE 英文网址：http://www.kde.org/
- Kate 编辑器网址：http://kate-editor.org/

Kate 作为 KDE 桌面环境的一部分，包含在软件包 KDE-Base（KDE 基本组件）中。作为 KDE 桌面环境中自带的文本编辑器，Kate 文本编辑器拥有许多优点。

- 文档管理功能：Kate 文本编辑器中提供了文档管理、浏览文件系统等功能。
- 文档搜索功能：用户可以使用文本内容搜索功能，查找包含目标文本、正则表达式的文件。
- 编辑器终端功能：用户可以随时从编辑器中调用系统终端。
- 多种特殊文本语法高亮：在 Kate 编辑器中可以设置多种语法高亮，其中包括编程语言（例如 C、C++等）、脚本编程语言（例如 Bash、Perl 等）、配置文件（例如 Apache 配置文件、文件服务器 Samba 配置文件等）等。
- 利用 Kate 编辑器自带的相关功能，还可以将其作为一个脚本程序设计和调试的环境等。

除此之外，Kate 还提供将当前文档作为邮件内容发送等功能，这些功能都得益于 Kate

编辑器与 KDE 桌面环境中的软件的无缝结合。

14.3.2　Kate 基本界面

由于 Kate 编辑器包含在 KDE-Base 基本组件套中，因此如果系统中安装了 KDE 桌面环境，就可以直接使用 Kate 文本编辑器。本小节将简单介绍如何启动 Kate 编辑器及其基本界面。

1．启动Kate文本编辑器

在 KDE 桌面环境中启动 Kate 文本编辑器有两种方式，第 1 种方式为在终端中使用命令 kate 启动。第 2 种方式为依次单击 K Menu（即桌面左下角小红帽图标）→Utilities→Editors→Kate 启动。

启动 Kate 编辑器时，会提示用户选择会话名称，如图 14.18 所示。会话用于设置编辑器默认动作，例如可以设置某一会话启动时，默认打开常用的文本文件等。通常情况下，使用 Default Session（默认会话）打开即可。选中 Default Session，然后单击 Open Session，即可使用默认会话启动 Kate 编辑器。启动后的 Kate 编辑器工作界面如图 14.19 所示。

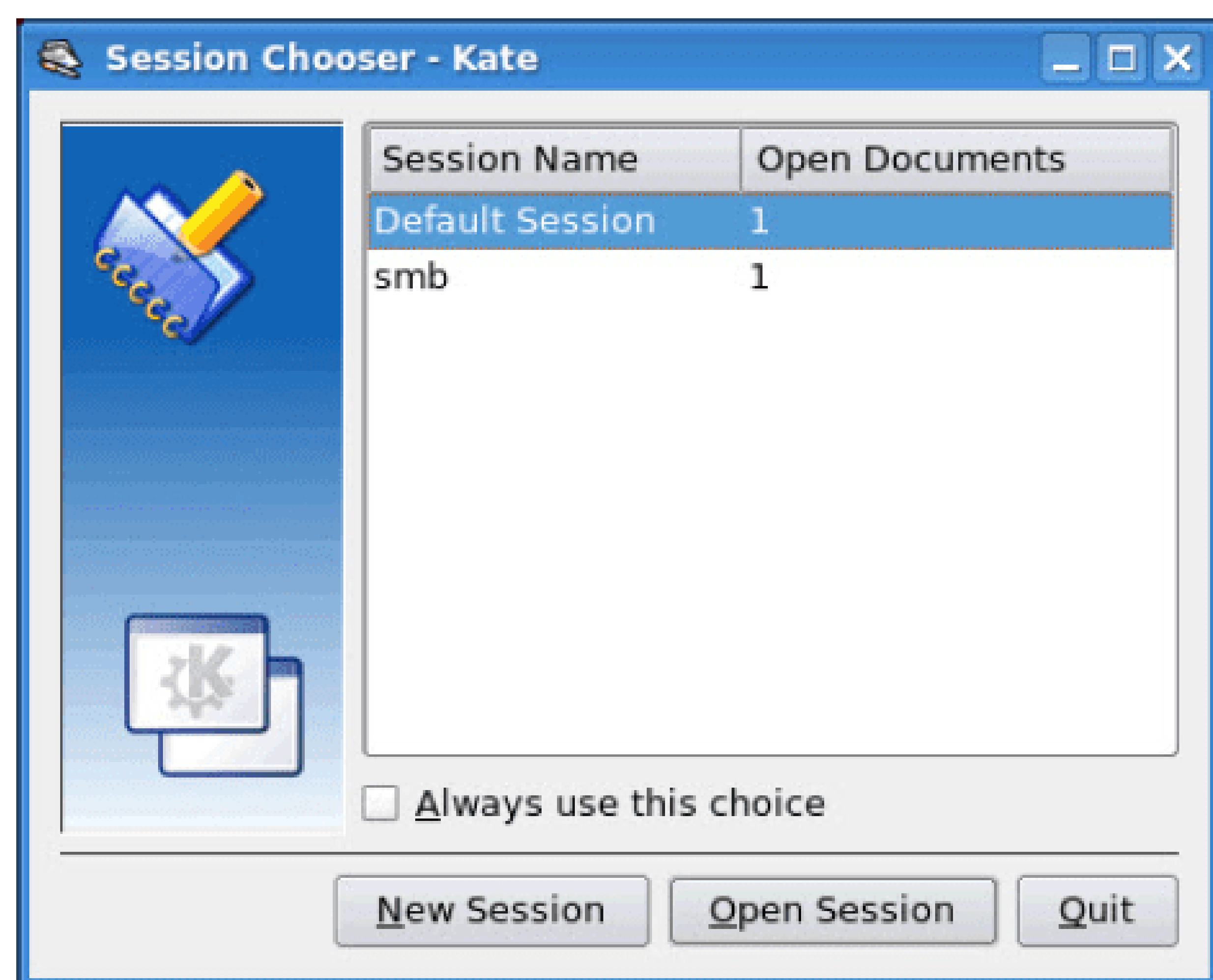


图 14.18　Kate 会话选择

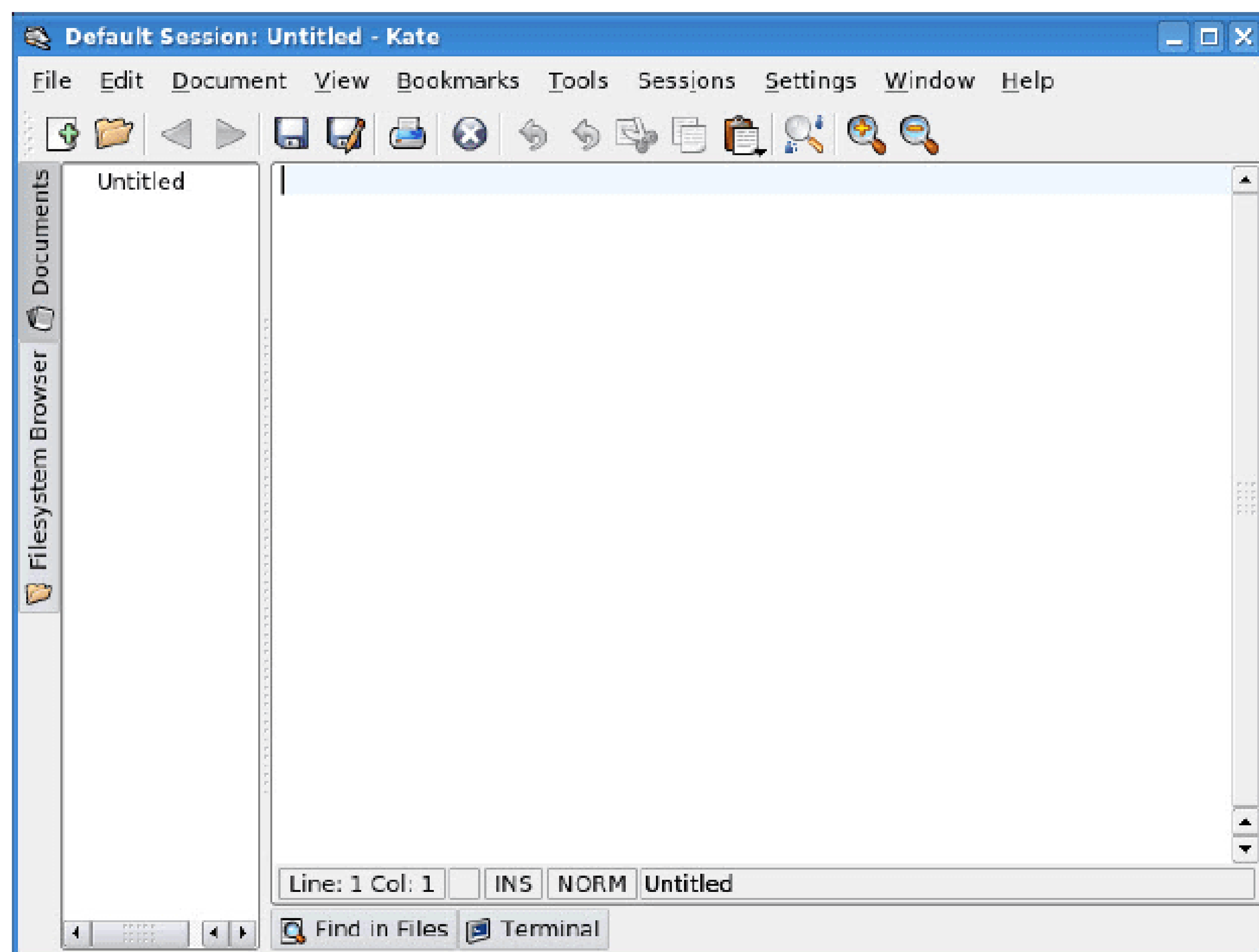


图 14.19　Kate 编辑器工作界面

注意：在不同的发行版中，使用第 2 种方式启动 Kate 文本编辑器时，菜单项及名称可能会不同。

2．Kate编辑器基本界面

Kate 编辑器界面与大多数图形界面中的应用程序非常相似，在顶部是标题栏、菜单栏、工具栏等。占据界面大部分的是文档管理器、文档浏览器及工作区，其文档管理器（即编辑器左侧的 Documents 标签页）可以在多个文本间切换编辑。而在文档浏览器中（即左侧的 Filesystem Browser 标签页），用户可以快速在文件系统上查找并打开需要编辑的文档。

与其他文本编辑器不同，Kate 的工作区最下方还存在 Find in Files 和 Terminal 两个缩

小的窗口，这两个窗口主要用于在文本中查找及打开终端。

14.3.3　快速移动光标

当用户编辑一个文本时，如果编辑器能够较快地移动光标至需要编辑的位置，可以大大提高编辑的效率。本小节将简单介绍在 Kate 中如何快速移动光标。

（1）快速移动光标

在 Kate 中，可以使用一些比较常规的方法快速移动鼠标，例如使用 Home、End 键快速移动光标至行首、行尾，使用 PageUp、PageDown 键向上、向下翻页等。当然也可以使用鼠标点击的方式移动光标。

（2）快速定位行

使用 Kate 作为编程环境的用户，可能希望能够快速定位到错误发生位置，由于在调试时通常都以行号的方式进行提示，因此需要快速定位行。在 Kate 编辑器中，要快速定位到行，可以依次单击菜单栏的 Edit→Go to Line，此时会弹出 Go to Line 对话框，如图 14.20 所示。

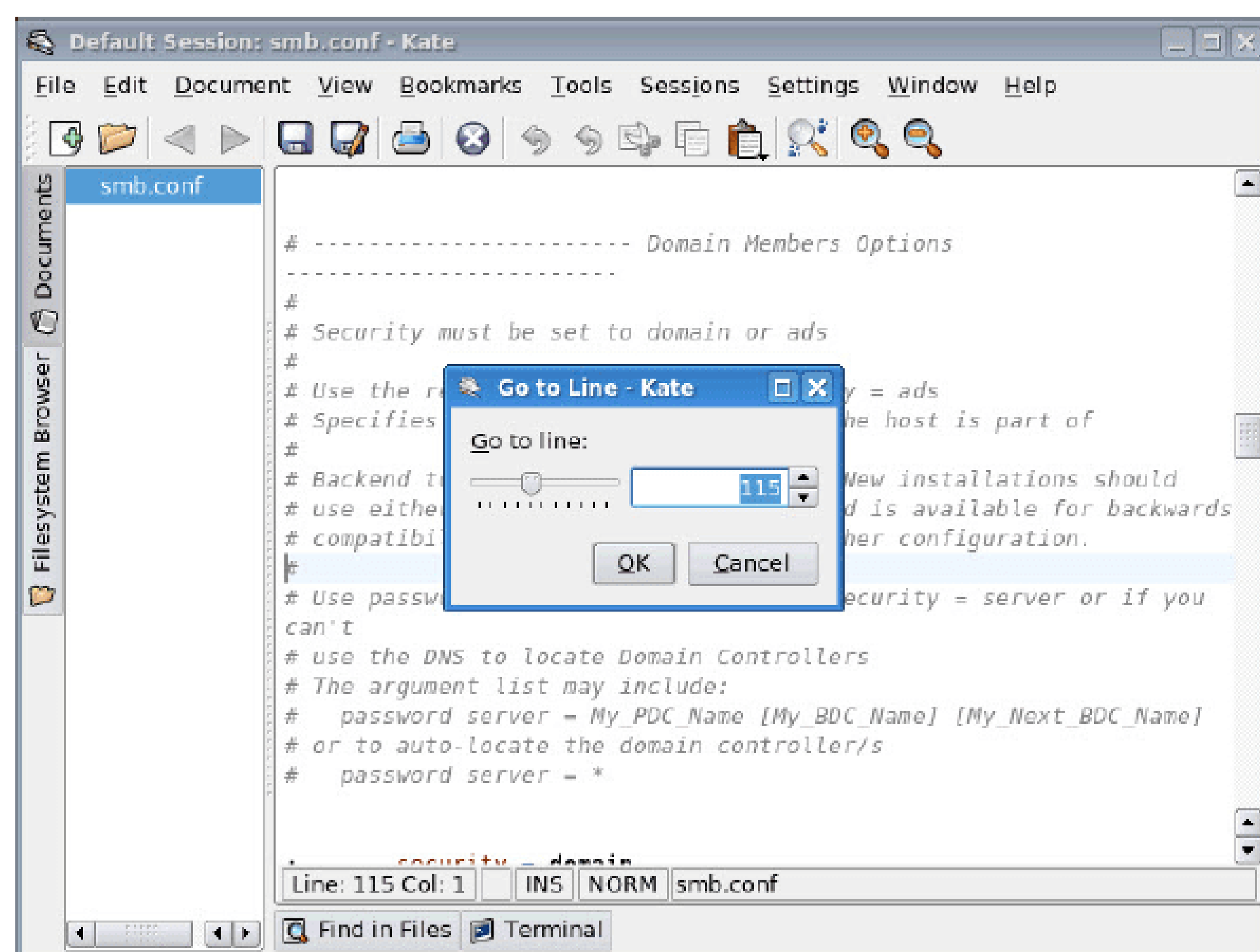


图 14.20　Go to Line 对话框

Go to Line 对话框中默认显示了当前光标所在位置及行号，用户可以在此对话框中输入要跳转的行号，单击 OK 按钮，即可跳转到指定的行。用户也可以使用滑块和对话框中的上、下箭头按钮指定行号。

14.3.4　查找和替换

许多编辑器都提供了查找和替换功能，Kate 也不例外。在 Kate 编辑器中，有三种类型的查找功能：查找、查找替换、查找文件。本小节将简单介绍 Kate 编辑器中的查找、替换等功能。

1．查找功能

要使用 Kate 编辑器的查找功能，可以依次单击菜单栏中的 Edit→Find，此时会弹出查

找对话框，如图 14.21 所示。

此时用户可以在 Text to find 文本框内输入要查找的文本，然后单击 Find 按钮在当前文本中查找。用户也可以选择 Regular expression 并单击 Edit 按钮，使用正则表达式进行查找。还可以使用 Case sensitive（区分大小写）、Find backwards（反向查找）、Whole words only（完全匹配）等选项配合查找。

如果查找到文本之后，需要继续查找下一个或者上一个匹配的文本，可以单击 Edit 菜单中的 Find Next 和 Find Previous 按钮。

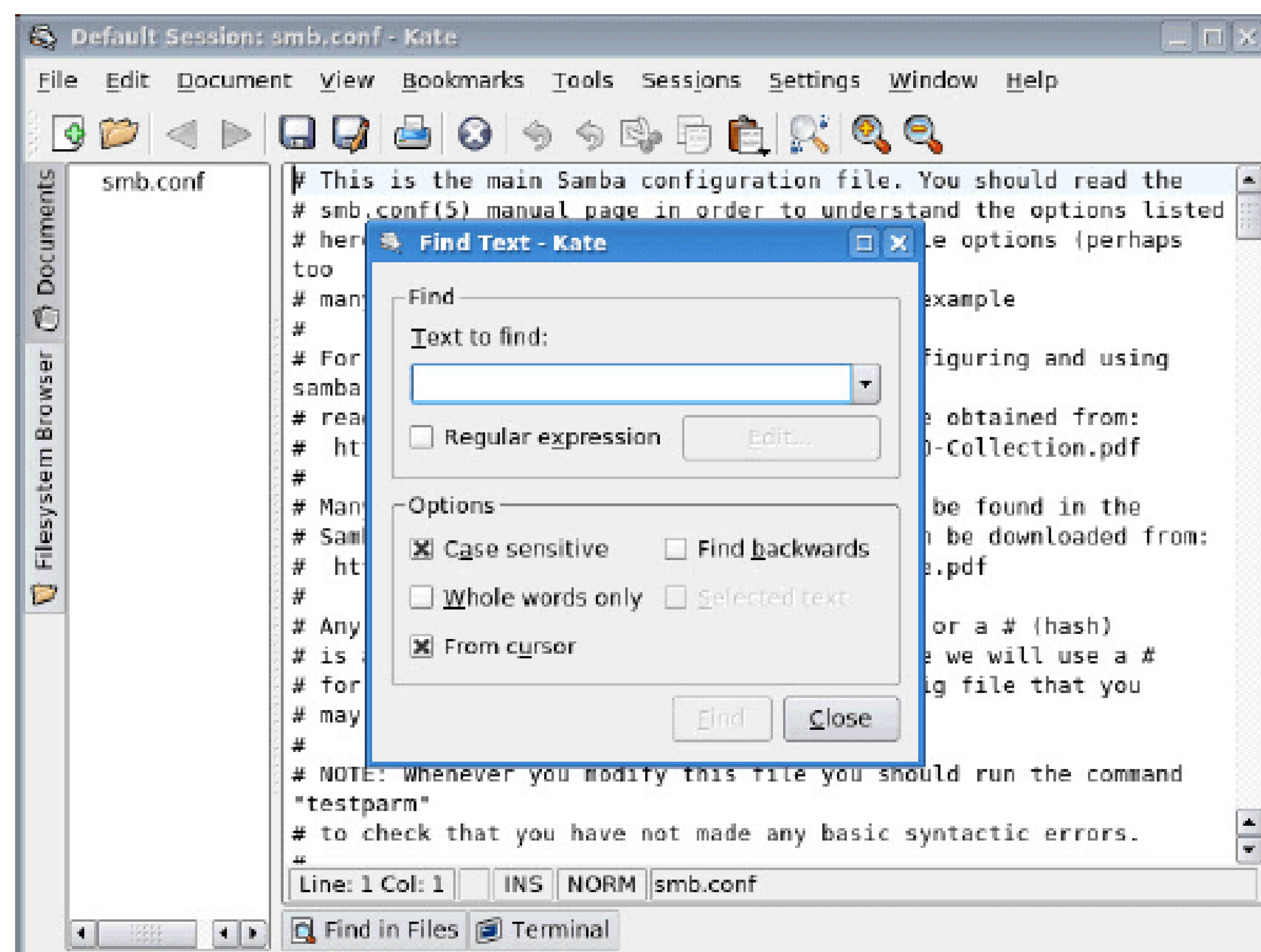


图 14.21　Find 对话框

2. 查找替换功能

要使用 Kate 编辑器中的查找替换功能，可以单击菜单栏中的 Edit→Replace，打开查找替换对话框，如图 14.22 所示。

此时用户可以在 Text to find 文本框中输入要查找的文本，在 Replacement text 文本框中输入替换后的文本，然后单击 Replace 按钮开始查找替换。

使用查找替换功能时，推荐用户选择 Prompt on replace。使用此选项时，Kate 编辑器会在替换时提醒用户，如图 14.23 所示。

如果确认需要替换当前文本，可以单击 Replace 按钮，如果当前查找到的文本是最后一个文本，可以单击 Replace & Close 按钮，替换后关闭。如果需要替换所有查找到的文本，可以单击 Replace 按钮，用户也可以单击 Find Next、Close 按钮，查找下一个文本和退出查找替换。

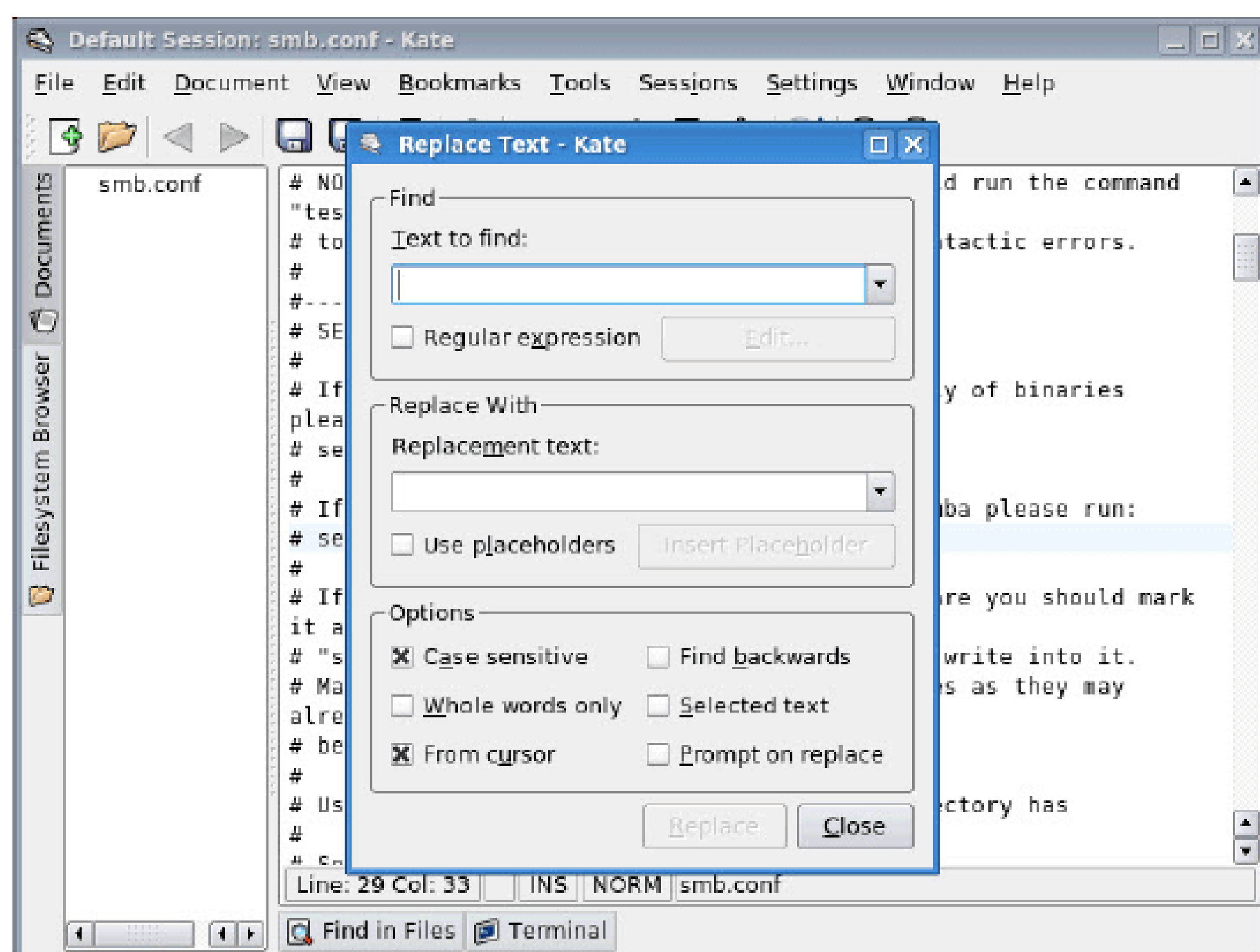


图 14.22　查找替换对话框

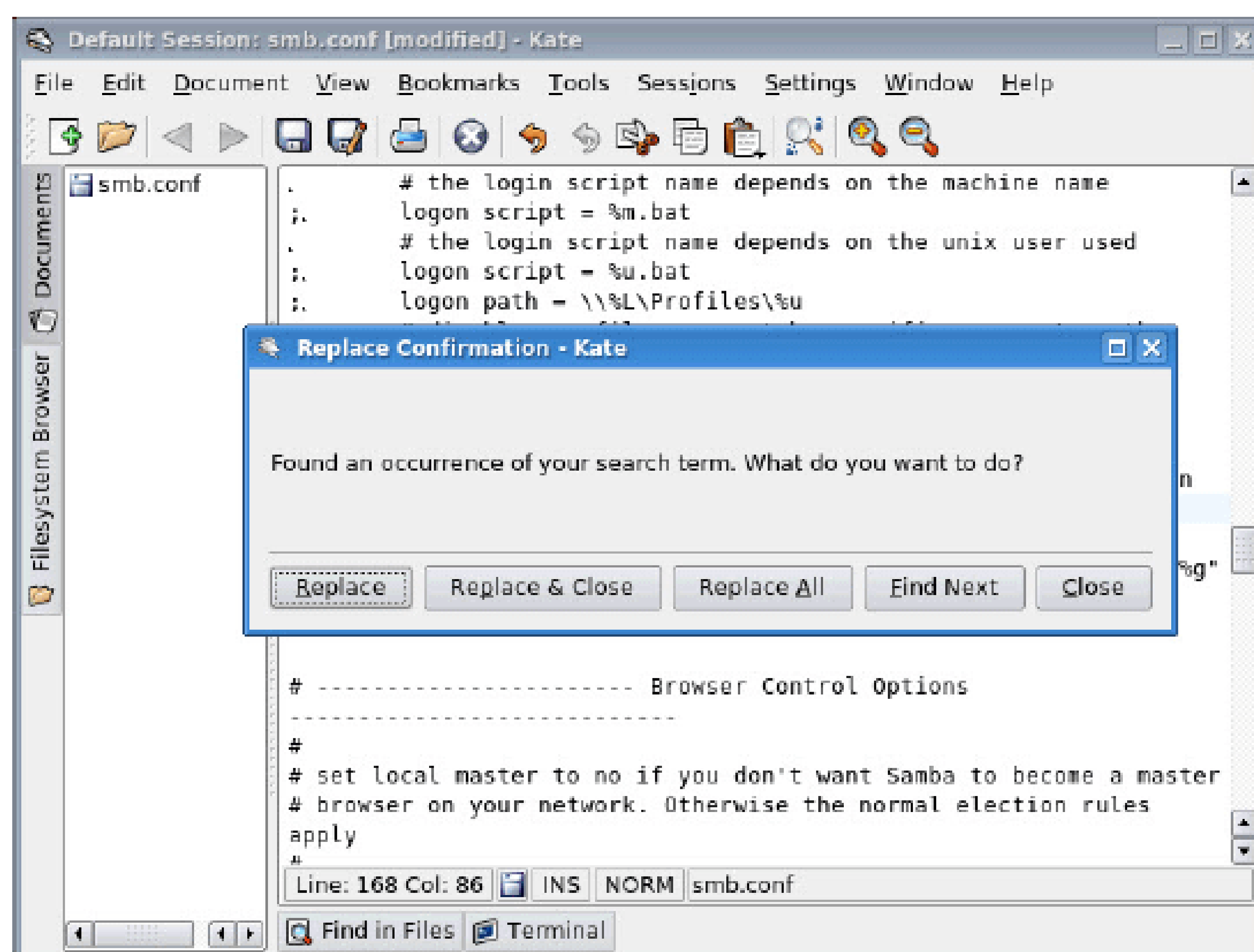


图 14.23　查找并提示用户

3. 查找文件

在 Kate 编辑器中，还提供了一个非常特殊的查找文件功能，这个功能允许用户在特定的目录中，查找包含指定文本内容的文件，即查找包含指定内容的文件。

要使用此功能，可以单击工作区底部的 Find in Files 按钮，此时将会打开 Find in Files

工具，如图 14.24 所示。

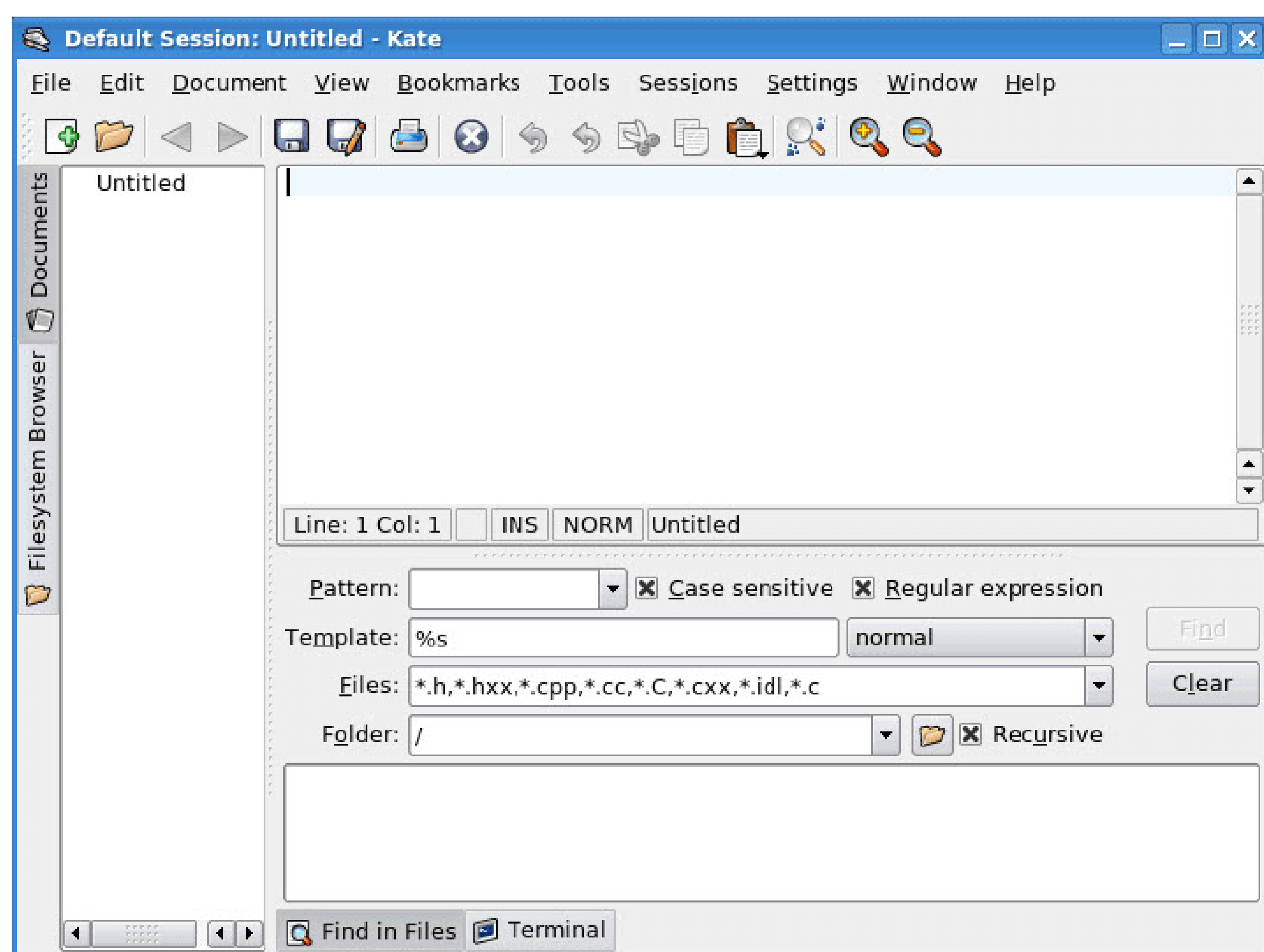


图 14.24　Find in Files 工具

用户可以在 Pattern 文本框中输入包含的文本，在 Folder 文本框内输入要查找的目录，单击 Find 按钮即可查找目录中包含指定文本的文件。用户也可以在 Template 中输入查找时使用的模板，在 Files 中指定查找的文件类型等。

14.3.5　拼写检查

拼写检查是一个非常实用的功能，它会检查用户输入的单词是否存在错误，并为错误单词提供了非常详尽的修改建议。本小节将简单介绍 Kate 中的拼写检查。

要使用拼写检查，可以依次单击 Tools→Spelling，此时将会弹出拼写检查对话框并自动检查，将检查到的错误显示在对话框中，如图 14.25 所示。

如果用户忘记错误的单词应该如何拼写，可以从 Suggested Words 中查看修改建议。查看修改建议后，用户可以按需要执行如下操作。

- 当前找到的并不是一个错误单词，可以单击 Add to Dictionary 按钮将当前单词加入到字典中。这种情况适合错误单词是一个特殊用法或该单词是 Kate 中不存在的单词的情况。
- 如果查找到的单词输入错误并且 Replace with 文本框中的单词正确，可以单击 Replace 按钮，使用正确的单词替换错误的单词。
- 如果确认查找到的单词是一个特殊的用法，可以单击 Ignore 按钮，忽略当前找到的单词。

用户也可以单击 Replace All 和 Ignore All 按钮，替换或忽略所有当前找到的错误单词。

提示：如果用户发现某些正确的单词也被提示错误，可以尝试在 Language 列表中选择使用的语言。

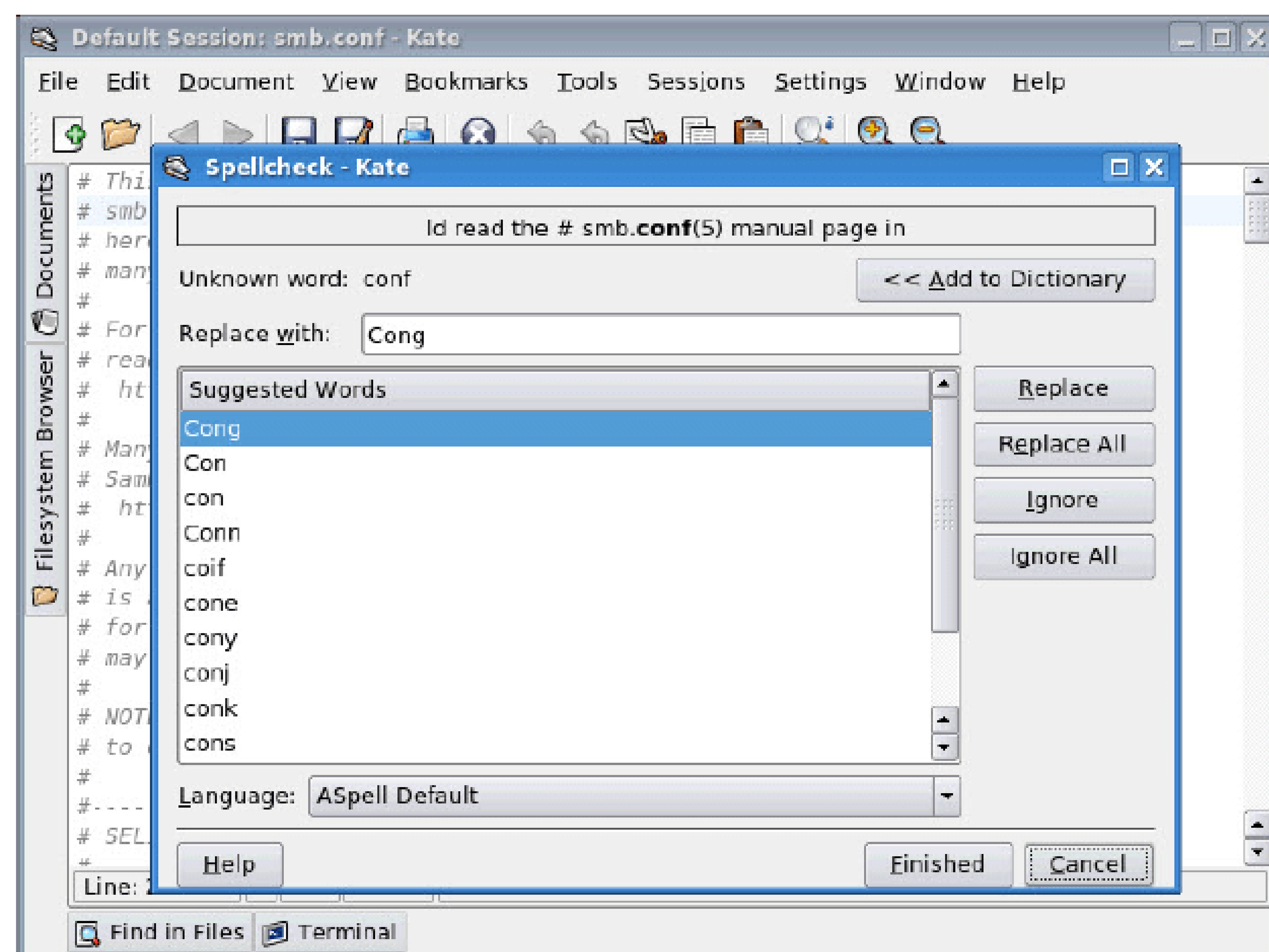


图 14.25　拼写检查

14.3.6　语法高亮

在 Kate 编辑器中，默认提供了多种文本类型的语法高亮，包括各种配置文本、脚本文件及程序源代码文件等。本小节中将简单介绍 Kate 编辑器的语法高亮功能。

要启用语法高亮，可以依次单击菜单栏中的 Tools→Highlighting，此时将会显示 Kate 编辑器中语法高亮可用的文本类型，按需要选择文本文件类型即可。在本例中依次单击菜单 Scripts（脚本）→Bash，Kate 就会使用不同的颜色标识文本，如图 14.26 所示。从中可以看出，Kate 使用不同的颜色标识不同的文本。这可以让程序设计者发现输入中的失误，大大方便了程序设计者编写程序。

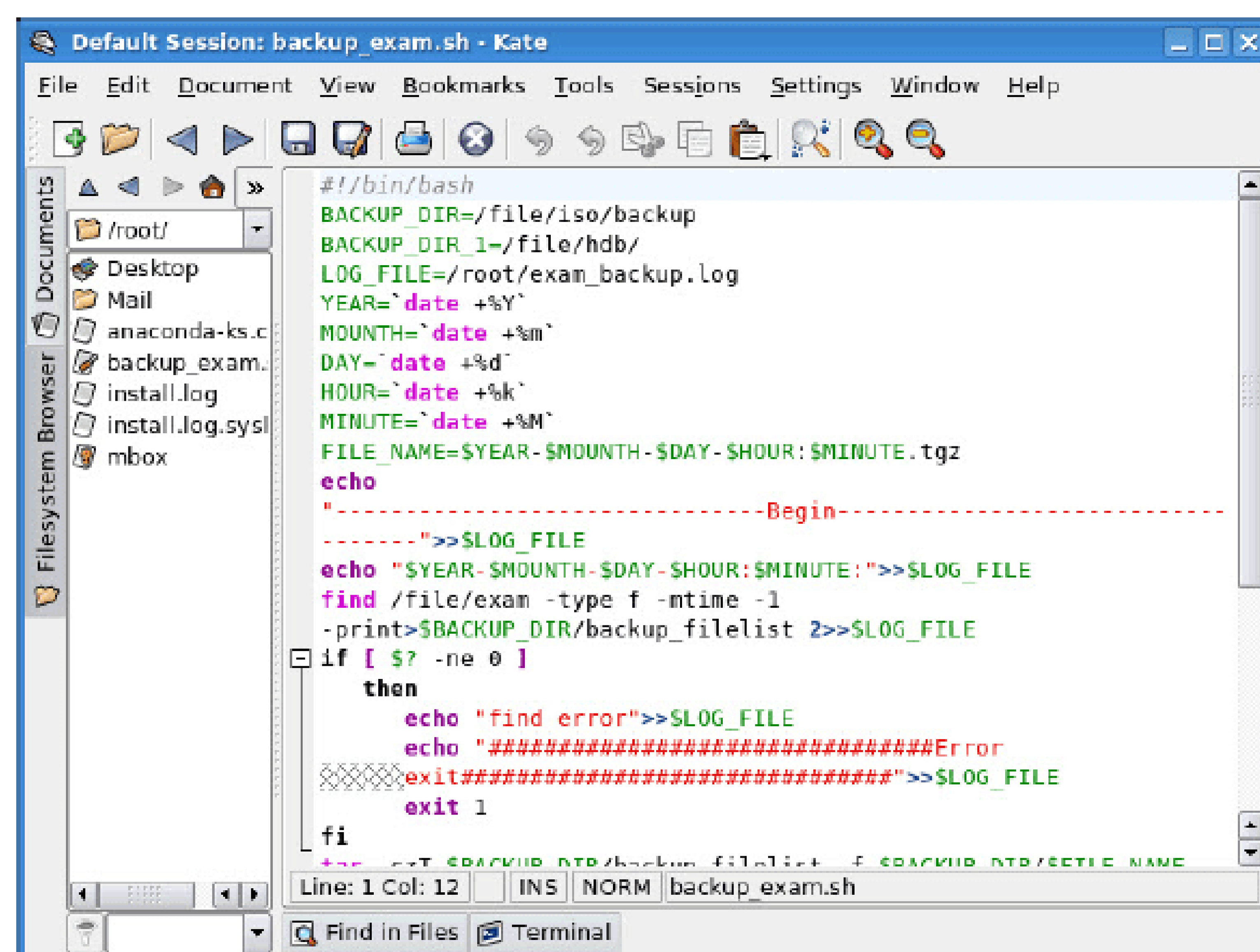


图 14.26　Kate 语法高亮

14.3.7　脚本编程环境

Kate 编辑器自带的许多工具集，构成了一个简单的脚本编程环境。初学者可以使用这个简单的编程环境学习脚本编程。本小节将简单介绍 Kate 自带的脚本编程工具。

（1）终端工具

许多时候用户在编写脚本时可能会忘记某个命令的用法，此时可以调出终端工具，在其中查看、验证命令的用法，然后写入脚本中。要调出终端工具，可以单击 Kate 编辑器窗口最下端的 Terminal，此时 Kate 编辑器将会调出终端工具，如图 14.27 所示。

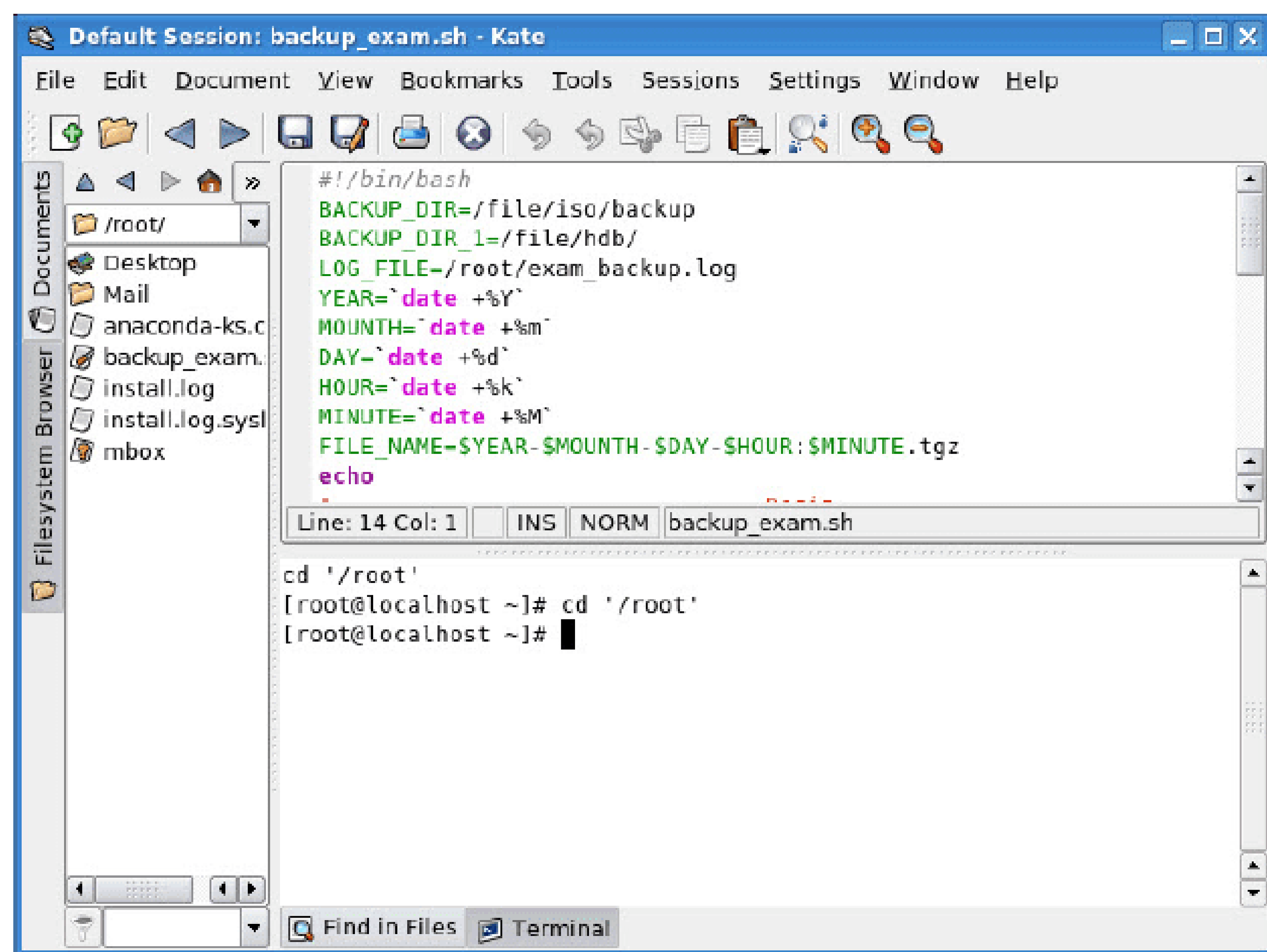


图 14.27　终端工具

此时用户可以在终端工具中查看命令的帮助信息、验证命令的用法是否正确等，然后将正确的命令写入脚本中。

有时可能用户需要验证某个已存在的脚本中相关语句的功能，此时可以先选中文件中需要验证的代码，然后依次单击菜单栏中的 Tools→Pipe to Console，Kate 会在终端中依次执行选中的代码。用户可以通过这种方式验证输入、代码功能是否正确。

提示：使用菜单栏中的 Pipe to Console 选项时，如果没有选中任何代码，Kate 会执行文本中的所有代码。

（2）代码折叠功能

使用 Kate 编写脚本时，可能需要在脚本中添加一些功能模块，这些模块可能是比较长的 if 语句，也可能是使用大括号{}包含的代码块。如果脚本中存在较多的功能模块，可能看起来会非常乱，这时可以使用 Kate 的代码折叠功能将功能模块收起。

要使用代码折叠功能，可以单击代码块前的减号“–”，此时“–”将会变为加号“+”，并将代码块折叠起来，如图 14.28 所示。

在图 14.28 中，第一个 if 语句处于未折叠状态，第二个 if 语句则处于折叠状态，读者可以对比二者，并区分代码是否处于折叠状态。

如果需要将折叠的代码打开，只需要单击代码前的“+”即可。

（3）执行编写的脚本

用户编写脚本之后，通常会执行脚本进行测试。在 Kate 中，如果需要执行编写的脚本，可以依次单击 Tools→External Tools→Run Script。这时 Kate 会执行当前编辑器中的脚本，由于执行过程可能一闪而过，因此用户需要在脚本中添加相关测试语句，以便于测试脚本是否正确运行。

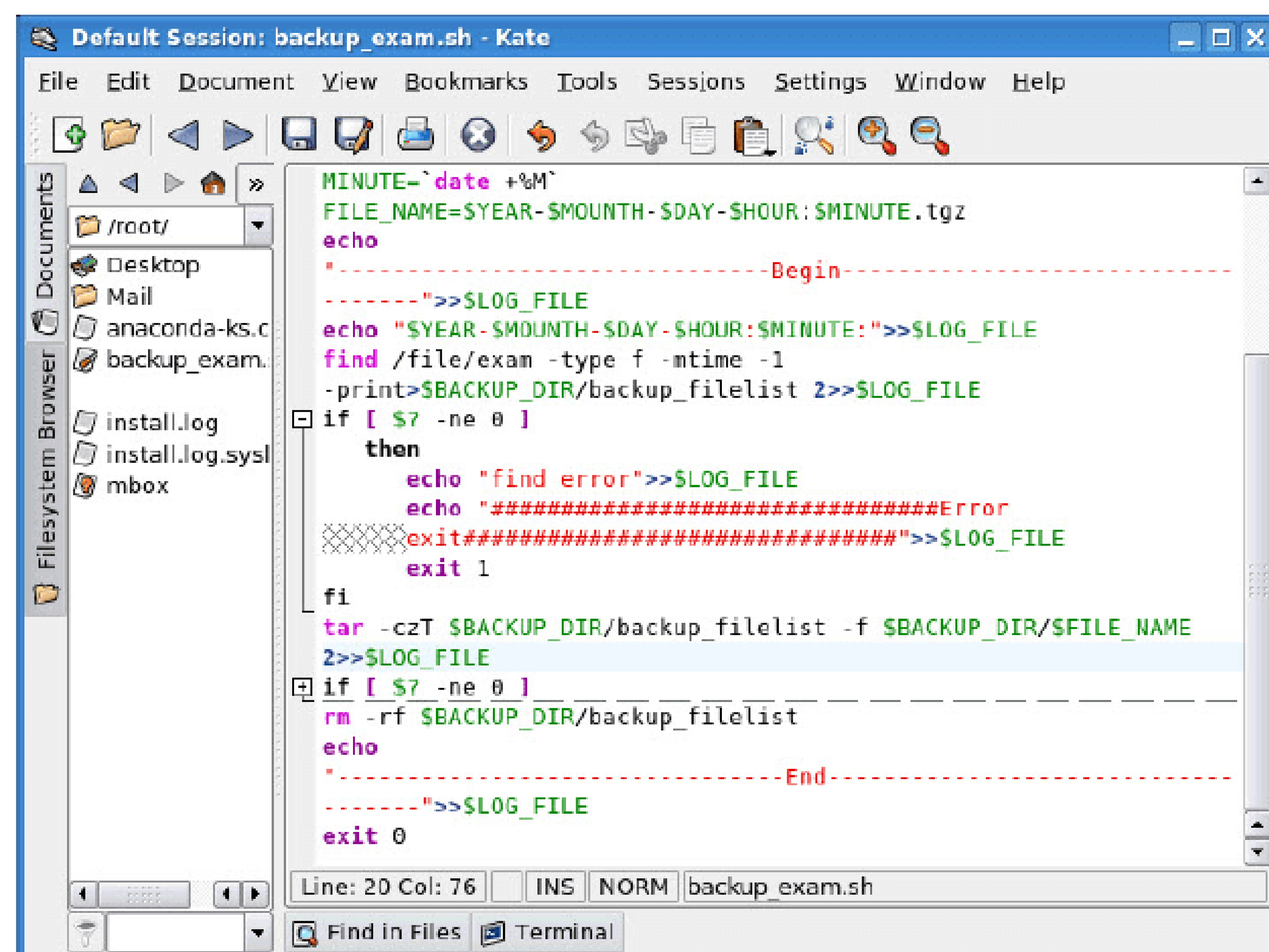


图 14.28　代码折叠功能

🔔**注意**：关于脚本编程及其相关方法将在本书的后续章节中介绍。

14.4　小　　结

- ❑ 14.1 节主要介绍了 Gentoo、Debian 中默认的文本编辑器 Nano。Nano 没有复杂的功能，也没有繁琐的操作，非常适合初学者使用。
- ❑ 14.2 节简单介绍了 Gnome 桌面环境中自带的文本编辑器 Gedit。Gedit 编辑器运行于图形界面中，拥有语法高亮、拼写检查、打印预览等功能，推荐使用 Gnome 桌面环境的读者使用。
- ❑ 14.3 节介绍了 KDE 桌面环境中自带的文本编辑器 Kate。Kate 与 Gedit 编辑器类似，都运行于图形环境中，也拥有语法高亮、拼写检查等功能。Kate 最大的特色是自带的工具与系统紧密融合，用户可以使用这些工具构建一个简单的脚本编程环境。

虽然大多数 Linux 系统都运行于字符模式下，但无论是在字符模式中，还是在图形界面中，都拥有非常不错误的文本编辑器。对于本章介绍的这些各具特色的文本编辑器，初学者可以按自己的实际需要选择使用。

第 3 篇　Shell 编程

第 15 章　Shell 脚本编程基础、变量

许多管理员都希望计算机能够智能而简单地处理一些简单的管理任务，例如让计算机自动完成备份等。普通用户则希望使用一条命令，计算机就能完成需要执行许多命令才能完成的任务，例如让计算机自动使用同一组命令处理多个报表。基于以上这些原因，Shell 脚本诞生了。

为实现某个任务，将许多命令组合后，写入一个可执行的文本文件的方法，称为 Shell 脚本编程。通常 Shell 脚本都是为了完成某项特殊任务而编写的，其中包括许多与任务相关的命令。从本章起将介绍 Linux 系统中的 Shell 脚本编程。

本章将详细介绍 Shell 脚本编程的基础知识，主要涉及的内容如下。

- Shell 脚本介绍及其使用范围。
- 如何创建一个 Bash 脚本，Shell 脚本的基本格式、组成要素等。
- 简单介绍 Shell 脚本文件的运行方式。
- 讲解如何获取用户输入、向脚本传递参数。
- 如何创建 Tcsh 脚本，Tcsh 与 Bash 脚本的区别等。
- Linux 系统中的变量概述、分类及如何使变量。
- Shell 脚本中的数组、数组的应用等。

说明：由于 Linux 系统使用 Bash 作为默认 Shell 且 Bash 的应用范围最广，因此除非特别说明，本书中的 Shell 脚本就是指应用于 Bash 环境的 Shell 脚本。

15.1　Bash 脚本编程基础

关于 Shell 的相关内容，在本书前面的章节中已经有所介绍。按照应用的 Shell 环境不同，可以将 Shell 脚本分为多种类型。其中最常见的是应用于 Bash 和 Tcsh 的脚本，通常将其称为 Bash 脚本和 Tcsh 脚本。

同 Tcsh 脚本相比，Bash 脚本在 Linux 中的应用相对较广。本节将简单介绍 Bash 脚本的特点、组成要素等基础知识，Tcsh 脚本将在下一节中介绍。

15.1.1　Shell 脚本概述

使用过 Windows 批处理脚本的读者，可能会对批处理脚本有深刻印象。将许多命令写入一个文本文件内，并将扩展名改为 bat，然后就可以运行批处理文件执行相关操作了。在 Windows 系统中，用户接触到最多、最常用的例子是：编写用于清除系统垃圾的脚本。读

者可以从互联网上下载一些批处理脚本自行研究，此处不做过多的介绍。

与 Windows 系统中的批处理脚本不同，在 Linux 操作系统中，无论是使用何种 Shell 编写的脚本，都可以使用变量、控制结构等比较复杂的结构。因此从结构上讲，Shell 脚本比批处理脚本要复杂很多。也因为如此，Shell 脚本的功能远比 Windows 中的批处理文件要强大得多。

Linux 系统中的 Shell 脚本通常具备以下特点。

- Shell 脚本将要运行的命令、语句写入一个文本文件中。文本文件可以使用 Vi 等文本编辑器编辑。
- 与其他编程语言不同，Shell 编程不用过多的训练，只需要熟悉系统中的命令、Shell 脚本的基本格式及相关语句就可以编写简单的 Shell 脚本。
- 与 Windows 等批处理脚本不同，Shell 脚本的功能几乎涵盖了系统的方方面面。例如管理应用程序、系统任务调度等，都可以交给 Shell 脚本处理。
- 与 C 语言类似，Shell 脚本的执行顺序也是从上而下，顺序执行的。
- Shell 脚本通常不需要特殊处理，对源代码文件加上可执行权限，就可以像应用程序一样执行。
- 与 C 语言、C++、C#等编程语言不同，Shell 脚本不需要编译、连接及生成可执行文件，直接由相应的解释器解释执行即可。
- Shell 脚本编辑与其他编辑类语言不同，在编写过程中不需要注重算法，只需要将命令进行有机的组合即可。

Linux 系统中的 Shell 脚本编程是每个系统管理员必备的技能之一，因此初学者应该掌握一门 Shell 脚本编程。

15.1.2　Shell 脚本的基本内容

Shell 脚本其实就是文本文件，因此建立新的脚本文件时，可以使用 Vi、Emacs、Nano 等文本编辑器。本小节将使用一个实例简单介绍如何新建 Shell 脚本，以及 Shell 脚本包括的基本内容。

在本例中，我们将在当前目录中新建一个名为 Hello.sh 的脚本文件：

```
#使用 vi 编辑器在当前目录中新建一个名为 Hello.sh 的脚本文件
# vi Hello.sh
```

然后进入 vi 编辑器的插入模式，输入脚本文件的内容。实际操作过程中，读者也可以使用自己熟悉的文本编辑器新建脚本文件。

提示： Shell 脚本同 Linux 系统中的其他文件一样，可以不使用扩展名。但为了方便识别，通常建议 Bash 脚本文件名以 sh 结尾，Tcsh 脚本文件以 csh 结尾。

本例中脚本文件的内容如下：

```
#使用 cat 命令查看 Hello.sh 的内容
# cat Hello.sh
#!/bin/bash
```

```
#This is a test script.
#This script will output Hello.
#5/6/11

echo "Hello !"
```

这是一个非常简单的脚本文件（通常称为 Hello 脚本，被许多编程语引用）文件，下面将以这个文件为示例讲解 Shell 脚本的基本格式。

1．调用Shell

本例中脚本文件的第 1 行内容是：

```
#!/bin/bash
```

这一行的内容用于告诉系统应该使用何种 Shell 来执行这个脚本，或者使用哪种 Shell 来解释执行这个脚本中的内容。本例中使用的是 Bash。

注意： 调用 Bash 语句也可以写为#!/bin/bsh，并且调用 Shell 的语句只能出现在脚本文件的第 1 行。

2．脚本注释

示例脚本的后几行内容：

```
#This is a test script.
#This script will output Hello.
#5/6/11
```

除第 1 行以外，脚本中所有以“#”开头的行都是注释。注释的主要作用是为了方便阅读和维护脚本，实际执行时系统会忽略注释。

【注释的内容】

编写脚本时，应该为脚本添加非常详细的注释内容，以便日后阅读和维护脚本。这些注释信息通常应该包括如下内容。

- 详细说明脚本文件的功能。
- 脚本文件建立的时间和修改该脚本文件的时间。
- 重要的语句块、复杂结构的作用。
- 脚本文件的作者、修改该脚本的作者。

为脚本添加注释信息时，应该尽可能详细，以便于后期阅读和维护。

【添加注释的注意事项】

添加注释时，需要特别注意以下几点。

- 除非将“#”放入引用符号内，否则符号“#”到行尾的所有内容都是注释内容。
- 为脚本添加注释时，应该将注释单独放在一行（正因为脚本中的注释通常都以单独的行为单位，许多书籍将注释称为注释行）。
- 如果使用的注释内容过长，可以将其分别写在多行中，并在每一行的开头都加上“#”。

注释在脚本维护过程中非常重要，因此任何情况下都建议为脚本添加详细的注释。

2. 脚本内容

脚本内容是脚本中最重要的组成部分，示例文件的最后一行就是脚本内容：

```
echo "Hello !"
```

脚本内容是实现脚本功能的一组命令的集合，由一个或多个命令组成。在较为复杂的脚本中，又将脚本内容划分为定义部分和主体部分。

- 定义部分主要用于定义脚本捕获的系统信号、使用的变量、函数和文件等。
- 主体部分的语句主要调用定义部分中的变量、函数，以实现脚本的功能。

在本例中，脚本内容只有主体部分，并且主体部分是一个 echo 命令，其功能是将字符串“Hello !”输出到标准输出。

【添加脚本内容的注意事项】

为脚本添加内容语句时，需要注意以下几点。

- 为了便于阅读，脚本文件的每一行只书写 1 个命令。
- 如果要在一行书写多个命令，可以用分号“;”分隔，但通常不推荐这样做。
- 在设计脚本内容时，应该将多个实现某一特定功能的命令写成一个功能模块或函数。
- 对于一些特殊的语句（例如流控制语句等），应该使用特殊的缩进格式以便于阅读。

一个合格的脚本文件，应该同时使用 Shell 调用、脚本注释，及脚本内容语句。为了使脚本的可读性更高，通常应该在 Shell 调用、脚本注释、脚本内容语句、功能模块和函数之间使用空行分隔。

技巧：与其他编程语言相比，脚本编程追求简单、实用。因此编写脚本时，不必面面俱到，使用最简单的语句进行堆砌实现脚本功能即可。

15.1.3　脚本的运行方式

编写一个脚本文件的目的，是为了能够让脚本能够正确运行，实现编写脚本的目的。脚本的运行方式通常有 3 种：使用 bash 命令、使用点号和设置脚本的执行权限，本小节简单介绍这 3 种方式。

1. 使用bash命令执行脚本

使用 Bash 编写的脚本，通常可以使用 bash 命令解释执行脚本：

```
#使用 bash 命令解释执行脚本 Hello.sh
# bash Hello.sh
Hello !
```

使用 bash 命令执行脚本时，系统会使用 bash 命令来解释并执行脚本中的每一行。

2．使用点号“.”执行脚本

许多时候可以使用“.”点号来执行脚本文件，这种执行脚本的方式通常用于调用系统脚本文件（系统脚本文件在第 16 章中介绍）。

使用“.”号执行示例脚本文件：

```
#使用点号执行示例脚本文件 Hello.sh
# . Hello.sh
Hello !
```

使用这种方式执行脚本文件时，系统会使用当前 Shell 解释执行脚本文件。因此如果当前 Shell 是 Bash，也可以使用这种方式执行脚本。

注意：使用 bash 命令和点号执行脚本时，可以省略脚本文件中的 Shell 调用语句。但如果脚本不是系统脚本文件，通常不建议这样做。

3．设置脚本为可执行

这种方式需要先为脚本文件添加可执行权限，然后就可以像应用程序那样执行脚本文件。

要运行示例脚本文件，应该先为脚本文件添加可执行权限：

```
#为脚本 Hello.sh 添加可执行权限，并验证
# chmod u+x Hello.sh
# ls -l Hello.sh
-rwxr--r-- 1 root root 93 Nov  7 21:24 Hello.sh
```

脚本文件具有可执行权限后，就可以像应用程序那样执行了：

```
#执行脚本文件 Hello.sh
# ./Hello.sh
Hello !
```

使用这种方式执行脚本文件时，如果脚本文件没有位于环境变量中的搜索路径，应该使用绝对路径或相对路径指定脚本文件的位置。

除了以上 3 种执行脚本文件的方式外，还可以使用 sh 命令执行 Bash 脚本文件，此处不再一一赘述。

15.1.4　接收用户输入

许多时候需要获取用户的输入，例如需要用户输入名称、脚本下一步执行的操作等。这时可以使用第 3 章中介绍的 read 命令，将用户的输入保存在变量中，然后再读取变量获取用户的输入。

下面是一个获取用户输入的示例脚本文件：

```
#查看示例脚本文件 example.sh 的内容
# cat example.sh
```

```
#!/bin/bash

#This is a sample script file.
#Accept user input for demonstration.
#5/6/11

#使用 read 命令将用户的输入保存到变量 NAME 中
echo -n "Input your name:"
read NAME
#使用 echo 命令输出变量
echo "Hello,"$NAME"."
```

运行上述示例脚本：

```
#为示例脚本添加可执行权限
# chmod u+x example.sh
#执行示例脚本 example.sh
# ./example.sh
#在提示信息后输入姓名
Input your name:Jhon
Hello,Jhon.
```

上面的脚本文件中，先使用 echo 命令输出提示信息，然后使用 read 命令将用户输入保存在变量 NAME 中。最后再使用 echo 命令将保存到变量中的用户输入输出到标准输出。

第 3 章中介绍了接收用户输入命令 read 的多种用法，读者可以参考示例脚本，自行编写几个小脚本熟悉脚本编程的方法。

15.1.5　向脚本传递参数

脚本文件在执行时，也可以像命令和应用程序一样，接收脚本参数。为了捕获向脚本传递的参数，可以使用系统定义的位置变量（关于变量的更多详细内容将在 15.3.2 小节中介绍）。

位置变量是一类比较特殊的变量，引用脚本参数时，可以使用$1 到$9 这 9 个变量。

下面是一个关于位置变量的示例脚本文件：

```
#使用 cat 命令查看示例脚本文件 example1.sh 的内容
# cat example1.sh
#!/bin/bash

#This is a sample script file.
#Demonstrate the use of position variables.
#5/6/11

#捕获各个位置变量并将其输出
echo "The first parameter:"$1
echo "The second parameter:"$2
echo "The third parameter:"$3
echo "The fourth parameter:"$4
echo "......"
```

上面的脚本文件中，分别使用$1、$2、$3、$4 捕获传递给脚本文件的第 1、2、3、4 个参数，并将其输出。

执行脚本文件：

```
#为脚本文件添加可执行权限
# chmod u+x example1.sh
#为执行的脚本文件传递 4 个参数：One、Two、Three、Four
# ./example1.sh One Two Three Four
The first parameter:One
The second parameter:Two
The third parameter:Three
The fourth parameter:Four
......
```

从上面的输出可以看出，位置变量$n 中保存着传递给脚本的参数。从$1 到$9 用于保存向脚本文件传递的第 1 个至第 9 个参数。

在前面几章中介绍了许多命令及其用法，此时读者可以参考示例脚本的格式，自行编写小脚本以熟悉脚本的基本格式和用法。

注意：使用位置变量通常只能获取到 9 个传递给脚本的参数，关于捕获更多参数的内容将在第 17.2 节中介绍。

15.2　Tcsh 脚本编程

Tcsh 广泛存在于各种类 UNIX 系统中，同时也是 Free BSD（源自加州大学伯克利分校的 UNIX 系统）等操作系统默认使用的 Shell。正因如此，Tcsh 脚本也广泛存在于许多操作系统中，例如 RHEL5.3 的别名设置目录/etc/profile.d 中就包含了许多 Tcsh 脚本。本节将简单介绍 Tcsh 脚本编程的基本情况。

15.2.1　输出字符串 Hello 的示例脚本

Tcsh 脚本的基本格式、编写方法及脚本中使用的命令等，与 Bash 脚本完全相同，只需要直接套用即可。本小节仍然以 Hello 脚本为例介绍 Tcsh 脚本编程。

新建一个名为 Hello.csh 的文本文件，其内容如下：

```
#使用 cat 命令查看示例脚本 Hello.csh 的内容
# cat Hello.csh
#!/bin/tcsh

#this is a test script.
#this script will output hello.

echo "Hello!"
```

在上面这个示例脚本中，第 1 行是调用 Tcsh 的 Shell 调用语句，后面几行以“#”开

头的是脚本注释，最后一行是脚本内容语名。

执行这个示例脚本：

```
#为脚本 Hello.csh 添加执行权限，并执行脚本
# chmod u+x Hello.csh
# ./Hello.csh
Hello!
```

对比 15.1.2 小节中的示例脚本 Hello.sh，读者可以发现 Tcsh 脚本的内容与 Bash 脚本的内容除 Shell 调用外，其他语句完全一致。这主要是因为在相同的系统中，虽然使用的 Shell 不同，但调用的命令文件都是相同的。

15.2.2　Tcsh 与 Bash 脚本的区别

在上一小节中，引入了一个简单的 Tcsh 脚本。虽然在简单的示例脚本中，Tcsh 脚本与 Bash 脚本相同，但实际上，一些较复杂的脚本还是存在许多差异。

- 在 Tcsh 中为变量赋值时，需要使用 set 命令。例如 set NAME=“Jhon”。
- 在 Tcsh 中数组的下标是从 1 开始的，而 Bash 中则是从 0 开始的。
- Bash 中设置全局变量（也称环境变量），使用的命令是 export，而 Tcsh 则使用 setenv。
- 获取脚本参数时，Bash 中一般使用$1、$2、$3 等位置变量，而 Tcsh 则使用$argv[1]、$argv[2]、$argv[3]等读取参数。
- 在 Bash 中获取信号的命令是 trap，而 Tcsh 中则使用 onintr。

除此之外，Tcsh 与 Bash 还存在许多区别，例如 Tcsh 中可以使用 goto 语句执行无条件跳转，但 Bash 不支持等，此处不再一一介绍，感兴趣的读者可以通过阅读相关文档了解具体内容。

由于 Tcsh 脚本主要用于 Free BSD 等 UNIX 系统中，因此本书中的脚本大多使用 Bash 作为环境，读者可以参考相关文档将这些脚本修改为 Tcsh 脚本。

15.3　Shell 中的变量

脚本编程过程中，必然会接触到 Shell 中的变量。可以将变量理解为一个能引用的“容器”，当需要使用“容器”中的内容时，直接引用“容器”即可。本节将简单介绍 Shell 中的变量。

注意：与其他编程语言不同，Bash 中的变量没有数据类型的区别。变量中的值都是以字符串的形式保存的，如果要进行数值计算，需要进行特殊转换。

Linux Shell 下的变量按其使用目的可以分为 3 种类型。

- 环境变量：用于保存操作系统运行时使用的环境参数。
- 位置变量：Bash 将传递给脚本的参数保存在位置变量中，以便于在脚本中引用这些参数。
- 预定义变量：由系统保留和维护的一组特殊的变量，这些变量通常用于保存程序

运行状态等。
- 自定义变量：由用户自行定义的变量。可用于用户编写的脚本、多个命令间的值传递等。

上面的变量中，除了自定义变量外，其他变量都是由系统自动生成并维护的。如非必要，用户不要轻易改变由系统生成和维护的变量值。本节将简单介绍这些变量及其使用方法。

15.3.1　保存系统运行情况的环境变量

系统环境变量是用户登录系统时，由系统自动生成并设置的一组变量。所有的进程、脚本都可以引用这些变量，因此环境变量的值通常与系统息息相关。本小节将简单介绍 Linux 系统中常用的环境变量。

系统启动后会产生许多环境变量，用户可以使用 set 命令查看这些环境变量：

```
#使用 set 命令查看当前系统中的环境变量
# set
BASH=/bin/bash
BASH_ARGC=()
BASH_ARGV=()
……
```

上面的命令输出了系统中使用的变量及其值，其中大多数都是环境变量。有时也存在一些用户自定义的变量等。下面将简单介绍系统中常见的环境变量及其值。

1. 系统中常见的环境变量

系统中有许多环境变量，有些环境变量用于保存当前用户的使用环境，例如 USER、HOME、PWD 等都是比较常见的。此处将简单介绍这些环境变量的作用。

注意：在不确定环境变量作用的情况下，不要修改环境变量的值，以免带来不必要的麻烦。

（1）家目录位置变量 HOME

HOME 变量用于保存当前登录用户的家目录位置，这个变量的值是由系统用户文件 /etc/passwd 中的用户家目录字段定义的。查看当前用户的家目录如下：

```
#使用 echo 命令查看变量 HOME 的值
# echo $HOME
/root
```

（2）系统语言变量 LANG

LANG 变量用于保存系统当前使用的语言，如果要临时修改当前系统使用的语言，通常可以通过修改该变量的值来实现。查看当前系统使用的语言如下：

```
#查看变量 LANG 中保存的系统语言
#en_US.UTF-8 表示英语
# echo $LANG
en_US.UTF-8
```

临时修改系统语言可以重新指定此变量的值：

```
#修改环境变量 LANG 的值为 zh_CN.UTF-8
#zh_CN.UTF-8 表示简体中文
# LANG=zh_CN.UTF-8
```

这时当前终端用户（本例中为 root）使用的语言将会变为简体中文。

（3）交互程序变量 SHELL

SHELL 变量用于保存用户当前使用的 Shell。可以通过查看该变量值的方法，快速查看当前使用的 Shell：

```
#快速查看当前用户使用的 Shell
# echo $SHELL
/bin/bash
```

（4）命令搜索路径变量 PATH

PATH 变量用于保存当前用户使用的命令搜索路径。当用户输入命令时，系统会尝试在该变量保存的目录中查找命令文件。

查看当前用户使用的命令搜索路径：

```
#通过查看变量 PATH 值的方法查看当前用户的命令搜索路径
# echo $PATH
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sb
in:/bin:/usr/sbin:/usr/bin:/root/bin
```

在上面命令的输出中，PATH 使用冒号“:”将不同的目录分隔开。关于 PATH 变量的应用，在第 9 章中已经介绍过，此处不再赘述。

（5）主提示符变量 PS1

变量 PS1 保存了用户使用的主提示符，这个主提示符通常都使用了一个字符串表达式。查看当前用户使用的主提示符：

```
#查看当前使用的主提示符表达式
# echo $PS1
[\u@\h \W]\$
```

以上命令输出了当前用户的主提示符表达式，这个表达式的结果形如：[root@localhost ~]#。

用户通过修改变量 PS1 的方式自定义主提示符，可以用于定义主提示符的表达式及其含义如下。

- \u：当前使用系统的用户。
- \h：当前计算机名称。
- \w：当前工作路径。
- \d：当前系统的日期。
- \$：提示符，root 用户是井号“#”，普通用户是美元符号“$”。
- \H：完整的计算机名称。
- \t：24 小时制的时间格式。
- \v：当前 Bash 的版本。

例如修改当前用户的提示符：

```
[root@localhost ~]# PS1='[\d \H \t \v]\$ '
[Thu Nov 11 localhost.localdomain 16:03:36 3.2]#
```

以上命令修改了当前用户的提示符，这个修改立即就生效了。

（6）辅助提示符变量 PS2

变量 PS2 保存了用户当前使用的辅助提示符，要理解辅助提示符的作用，可以查看下面这个例子：

```
#查看当前使用的辅助提示符
# echo $PS2
>
#辅助提示符一般用于强制换行时，提示用户继续输入命令
#下面的>就是辅助提示符
# echo \
> Hello
Hello
```

在上面这个示例中，当用户使用强制换行符“\”时，系统使用变量 PS2 中保存的辅助提示符提示用户继续输入未完成的命令。

如果要修改辅助提示符，使用如下命令：

```
#将辅助提示符修改为<，并使用强制换行符验证结果
# PS2='< '
# \
<
```

从上面的命令输出中可以看到，辅助提示符修改后会立即生效。

许多时候，可以通过修改环境变量值的方法定制用户的工作环境。用户可以在命令提示符中临时修改环境变量的值，也可以通过修改配置文件的方法，永久地改变用户的工作环境。

2. 只读环境变量

除了前面介绍的环境变量之外，系统中还定义了特殊的只读环境变量。与其他环境变量相比，这些只读环境变量的值是无法更改的。

查看系统中的只读环境变量如下：

```
#使用 readonly 命令查看只读环境变量
# readonly
declare -ar BASH_VERSINFO='([0]="3" [1]="2" [2]="25" [3]="1" [4]="release"
[5]="i686-redhat-linux-gnu")'
declare -ir EUID="0"
declare -ir PPID="9064"
declare -r SHELLOPTS="braceexpand:emacs:hashall:histexpand:history:int-
eractive-comments:monitor"
declare -ir UID="0"
```

从上面的命令输出中可以看到，只读环境变量都是一些非常重要的变量。以 UID 变量为例：人们编写的脚本中，通常使用此变量的值判断用户的身份（值为 0 则表示 root 用户）。如果普通用户能够修改这个变量的值，就可以冒充其他用户，这是非常不安全的。

3．修改环境变量

前面已经介绍了在命令提示符中如何修改环境变量，但在命令提示符中修改的环境变量将在用户重新登录、系统重启后消失。这时可以在配置文件中修改环境变量。

在 Linux 系统中，用户环境变量的定义工作主要是在以下文件中完成的。

- /etc/profile：全局用户配置文件。如果修改了此文件中的设置，修改的设置将会影响系统中的所有用户。
- ~/.bash_profile：用户个人配置文件。如果修改了此文件中的设置，修改的设置只会影响单个用户。
- /etc/bashrc：全局环境变量配置文件。此文件中定义了所有用户的环境变量。
- ~/.bashrc：个人环境变量配置文件。此文件中定义了用户的环境变量。

由于全局用户配置文件和全局环境变量配置文件中的设置会影响所有用户，因此不推荐修改这两个文件。

此处以修改用户的主提示符为例，讲解修改用户环境变量的方法。修改用户个人配置文件.bash_profile，修改后的内容如下：

```
#查看修改后的文件.bash_profile 的内容
# cat .bash_profile
# .bash profile

……
#以上省略部分为原来的内容

#以下为新加入的内容
#modify PS1
PS1='\h: '
#使用 export 将变量 PS1 转换为全局变量
export PS1
```

文件最后定义了新的主提示符。修改完成后，用户只需要重新登录，环境变量 PS1 就可以生效。

上面这个示例修改的是个人用户配置文件.bash_profile，在实际操作过程中，也可以修改个人环境变量配置文件.bashrc。

提示：此处仅简单介绍了通过修改配置文件定制环境变量的方法，更多的内容将在第 16 章中介绍。

15.3.2　传递参数的位置变量

为了方便获取传递给脚本的参数，Bash 定义了 9 个位置变量，分别是$1、$2、$3、$4…$9。脚本编写者在脚本中引用这 9 个变量，获取传递给脚本的参数。

位置变量在本书的 15.1 节中已经介绍过，此处再给出一个简单的示例脚本 example2.sh 说明其用途：

```
#查看示例脚本 example2.sh
# cat example2.sh
#!/bin/bash

#This is a test script.
#5/11/11

#每 3 个一行，输出获取到的 9 个参数
echo '$1' "=" $1  '$2' "=" $2  '$3' "=" $3
echo '$4' "=" $4  '$5' "=" $5  '$6' "=" $6
echo '$7' "=" $7  '$8' "=" $8  '$9' "=" $9
```

运行示例文件：

```
#为示例文件添加可执行权限
# chmod u+x example2.sh
#运行示例文件并传递 9 个参数
# ./example2.sh a b c d e f g h i
$1 = a $2 = b $3 = c
$4 = d $5 = e $6 = f
$7 = g $8 = h $9 = i
```

脚本正确输出了传递给脚本的值。

使用位置变量时，需要注意位置变量只能获取前 9 个参数。关于在脚本中获取更多参数的方法，将在 17.2 节中介绍。

15.3.3　系统预先定义的变量

系统预先定义的变量简称预定义变量，是由系统预先定义的一组变量，这些变量通常用于保存与系统、命令等有关的信息。预定义变量由系统自动生成、维护，用户无须修改其值。本小节将简单介绍如何使用预定义变量。

常见的预定义变量及其含义如下。

- $0：保存当前程或脚本的名称。
- $*：保存传递给脚本或进程的所有参数。
- $$：当前进程或脚本的 PID 号。
- $!：后台运行的最后一个进程的 PID 号。
- $?：用于返回上一条命令是否成功执行。如果成功执行，将返回数字 0，否则返回非零数字（通常情况下都返回数字 1）。
- $#：用于保存脚本的参数个数。

在上面几个预定义变量中，$0 看起来非常像位置变量，但实际上它是预定义变量，初学者需要注意区分。

（1）为了讲解预定义变量的用途，此处引入一个关于预定义变量的示例脚本 example3.sh：

```
#查看示例脚本 example3.sh 的内容
# cat example3.sh
#!/bin/bash
```

```
#This is a test script.
#5/13/11

#利用$0 输出当前脚本的名称
echo "script name:"$0
#输出传递给脚本的所有参数
echo "all parameters:"$*
#输出脚本的 PID
echo "PID is the script:"$$
#输出上一条命令的执行状态
echo "success of the previous command:"$?
```

脚本的执行结果如下：

```
#执行示例脚本并传递 3 个参数
# ./example3.sh One Two Three
script name:./example3.sh
all parameters:One Two Three
PID is the script:10521
success of the previous command:0
```

上面的输出结果验证了预定义变量的含义。

（2）下面是一个预定义变量$!的示例：

```
#将两个进程放入后台执行，并使用变量$!输出最后一个进程的 PID
# sleep 100 &
[1] 10542
# sleep 300 &
[2] 10543
# echo $!
10543
```

以上面的命令输出中可以看出，预定义变量$!主要用于处理后台进程。由于在脚本中处理后台进程的情况较少，因此了解这个变量即可。

在 Linux 系统中编写管理脚本时，经常用到预定义变量，例如使用预定义变量判断命令是否正常执行，捕获脚本的参数等。这在许多时候是非常方便的，因此初学者应该掌握预定义变量的用法。

15.3.4　用户自定义变量

与其他编程语言类似，在 Bash 中也可以使用自定义变量，并且使用方法大同小异。唯一需要注意的是，Bash 中的变量没有数据类型（即任何变量都可以存放数字、字符串等）。本小节将介绍如何使用自定义变量。

【自定义变量的命名规则】

在为变量命名时，需要注意以下几点。

- 由于变量没有具体的数据类型，因此在定义变量（许多编程言中称为声明变量）时可以不必定义其类型，直接赋值即可使用。
- 在 Shell 中变量名称可以由大写字母、小写字母、下划线、数字等符号组成。

- 定义变量时，建议使用大写字母、下划线和数字组成变量名，以免引起不必要的误解。例如定义名为 mail 的变量，在执行、阅读时非常容易与命令 mail 混淆。

除以上注意事项外，还需要特别注意变量也对大小写敏感。例如 Mail 同 mAil 是不同的两个变量。

【变量的作用域】

变量的作用域是指可以变量的引用范围，也可以理解为变量在哪个区域起作用。Bash 中的变量作用域规则可以归纳为以下两点。

- 除非使用了 export 命令，否则在脚本中定义的变量（包括函数中定义的变量）的作用域是整个脚本。
- 在命令提示符中定义的变量和使用 export 定义的变量都是全局变量。全局变量可以在当前用户的任何脚本、命令中引用。

从以上两点可以看出，如果用户需要在脚本中定义全局变量，必须使用 export 命令。

【用法示例】

（1）定义一个名为 NAME 的变量，并为其赋值为 Jhon：

```
#直接为变量赋值即可声明变量
#变量的值可以不必放入引号内，除非值中含有特殊字符
# NAME=Jhon
```

执行上面的命令之后，系统会将字符串 Jhon 赋值给变量 NAME。

（2）如果用户需要经常使用一个变量，可以将这个变量定义为一个环境变量。例如要使用变量保存备份目录：

```
# BACKUP DIR=/file/backup
# export BACKUP_DIR
```

（3）如果用户希望定义一个不能更改值的只读变量，可以先定义变量，然后使用 readonly 命令将变量转换为只读变量。

例如要定义一个变量 PI：

```
#定义变量 PI
# PI=3.14
#使用命令 readonly 将变量 PI 转换为只读变量，然后验证
# readonly PI
# PI=3.1415
-bash: PI: readonly variable
```

从上面的示例中可以看出，试图修改只读变量的值时，Bash 会返回变量只读的错误信息。

（4）使用变量时，无论是引用变量，还是变量间的运算，都应该使用变量引用符“$”：

```
#定义变量 NAME，然后在 echo 命令中引用
# NAME=Jhon
# echo "Hello,$NAME."
Hello,Jhon.
```

下面是一个使用变量运算的例子：

```
#先定义变量 A 和 B 的值
# A=12
# B=12
#使用 expr 命令计算变量 A、B 之和、
# expr $A + $B
24
```

在上面的示例中，变量之间的数值运算使用了命令 expr。这是因为 Bash 中的变量默认都是字符串类型的，只有将其转换后才能运算。

（5）为了讲解用户自定义变量的使用方法，此处引入一个用于猜数游戏的示例脚本文件 example5.sh。在这个示例脚本中，先使用环境变量$RANDOM 产生随机数，然后使用 while 获取用户输入。判断用户输入的数值是否等于随机数，如果不等于就给出相应的提示，否则就提示用户猜对，并退出脚本。

查看示例脚本的内容：

```
#查看示例脚本文件 example5.sh 的内容
# cat example5.sh
#!/bin/bash

#this is a example
#5/14/11

#产生一个 10 以内的随机数
#使用取余的方法获取 10 以内的随机数
RAN=`expr $RANDOM % 10`
#由于产生的随机数可能是 0，因此需要加 1
RAN=`expr $RAN + 1`
echo "This is a guessing game."
echo "Number range:1-10"
#使用一个循环判断用户的输入
while true
  do
   #提示用户输入数字
   echo -n "Please enter the number you think:"
   read NUM
   #判断用户是否在 1～10 以内,如果不在 1～10 范围以内，则提醒用户重新输入
   if [ $NUM -lt 1 ] || [ 10 -lt $NUM ]
    then
     echo "You enter the number out of range."
     continue
   fi
   #判断用户输入与产生的随机数是否相等
   #如果用户输入与随机数相等，则输出提示信息，清除相关变量并退出
   if [ $NUM = $RAN ]
    then
     echo "You guessed it."
     unset NUM RAN
     exit 0
```

```
    fi
    #判断用户输入是否小于随机数，如果小于随机数，则产生相关提示
    if [ $NUM -lt $RAN ]
     then
      echo "You enter the number is too small."
    else
      echo "You enter the number is too large."
    fi
done
```

上面的示例脚本文件中，先使用环境变量 RANDOM 产生了一个 10 以内的随机数，然后使用循环接收用户的输入，并判断用户的输入是否正确。

提示：对于许多初学者而言，可能并不能看懂上面的示例脚本。学习完本书的后面几个章节后，就可以非常轻松地看懂这个脚本。

运行示例脚本：

```
#运行示例脚本，并验证其功能
# ./example5.sh
This is a guessing game.
Number range:1-10
#输入一个超出范围的值，验证脚本是否能正常判断
Please enter the number you think:11
You enter the number out of range.
Please enter the number you think:5
You enter the number is too large.
Please enter the number you think:1
You enter the number is too small.
Please enter the number you think:2
You guessed it.
```

这个示例脚本中运用到环境变量和自定义变量，很好地说明了变量及其使用。

（6）当变量完成其作用不再使用时，应该将变量从系统中清除，以释放其占用的系统资源。

清除变量可以使用命令 unset：

```
#使用 unset 清除变量，并验证
# echo $NAME
Jhon
# unset NAME
#由于变量 NAME 已经被清除，因此以下输出为空
# echo $NAME

```

需要注意的是，使用 unset 命令清除变量时，不必使用变量引用符。

技巧：如果用户编写了数个脚本，并且需要使用许多变量，可以为这些变量编写单独的变量文件，或将变量设置为全局变量，需要使用时引用这些变量即可。这样既可以方便脚本的编写，又方便维护变量。

15.4　Shell 中的数组

与许多编程语言一样，Bash 也提供了数组功能。数组是一类非常特殊的变量，与其他变量相比，数组不是单一的变量，而是一组相似的变量的集合。这些变量使用相同的变量名和索引（称为下标）来表示这个变量集合中的不同组成元素。本节将简单介绍数组的使用方法。

15.4.1　数组的定义

与使用变量一样，使用数组前也必须对其进行定义（许多编程语言将这个过程称为声明），定义数组时应该为其赋值。

【数组的格式】

完整的数组元素由数组名和索引组成，其格式如下：

```
ARRAY[index]
```

上面的数组元素的格式中，index 就是数组元素的索引（下标）。在 Bash 中，数组元素的索引（下标）是从 0 开始的。由于数组也是一种变量，因此定义数组时，数组的命名规则与变量相同。

【定义数据】

（1）在 Bash 中定义数组有两种方法，第一种方法是直接在定义时为所有的数组元素赋值：

```
#在定义数组时为其赋值
# ARRAY=(1 2 3 4 5)
```

使用这种方法对数组进行定义时，系统会自动为数组中的各元素进行赋值操作。例如上面的示例中，会自动为 ARRAY[0]赋值为 1、ARRAY[1]赋值为 2……依此类推，直到完成对所有数组元素的赋值操作为止。

（2）第二种方法是对数组中的每个数组元素进行赋值：

```
#为数组中的特定元素赋值
# ARRAY[0]=1; ARRAY[4]=5; ARRAY[3]=4
```

使用这种方法对数组定义时，可以不按照数组下标的先后顺序进行操作，也不必对数组中的每一个元素都赋值。

提示：使用第二种方法时，如果出现了某些数组元素没有赋值的情况，系统将会自动将一个空值赋值给这些元素。

（3）如果需要修改数组中某个元素的值，可以使用数组定义中的第二种方法：

```
#使用重新定义的方法修改数组元素的值
# ARRAY[1]=2
```

```
# ARRAY[2]=3
……
```

上面的命令修改了第 2、3 个元素的值（元素个数应该从下标 0 开始计算）。

15.4.2　数组的使用

由于数组中保存的是一组变量，因此其使用方法与变量不同。本小节将简单介绍如何使用组。

（1）如果需要引用数组中的某个元素，可以使用数组名称加索引的方式：

```
#使用 echo 命令显示数组 ARRAY 中第 3 个元素的值
# echo ${ARRAY[2]}
3
```

与其他语言中引用数组元素不同，在 Bash 中需要将数组元素放入大括号“{}”内。其目的是为了避免 Bash 将数组名误解为一个变量。

（2）如果不指定数组索引，将会显示数组中第 1 个元素的值：

```
#不使用下标时，将默认引用数组中的第 1 个元素
# echo $ARRAY
1
```

（3）有时可能希望看到数组中所有元素的值，这样做的目的可能是为了检查其值是否有错误：

```
#输出数组中所有元素的值
# echo ${ARRAY[*]}
1 2 3 4 5
```

在上面的示例命令中，“*”表示所有元素下标。

（4）当数组中的元素非常多时，可能查看起来非常不方便。这时可以指定查看的元素范围，例如：

```
#查看数组中下标大于等于 2 的所有元素的值
# echo ${ARRAY[@]:2}
3 4 5
```

（5）有时需要获得数组的长度，即数组中一共有多少个元素。这时可以使用以下形式：

```
#显示数组中的元素个数
# echo ${#ARRAY[@]}
5
```

上面这个示例命令输出的数字 5 表示数组中有 5 个元素，而非数组中的元素下标最大值。

（6）为了演示数组的使用，此处引入一个示例脚本 example4.sh。该脚本的作用是按数组元素的值从小至大排序。脚本先取得数组中的第 1 个元素值，然后与数组中的第 2、3、4…个元素进行比较。如果数组中的元素小于第 1 个元素的值，就将两个元素的值互换。这样直到最后一个元素，再取得数组的第 2 个元素的值进行比较，就这样直到数组中的最后

一个元素。比较完成后，数组中的元素的数值就从小到大排序完成了。

查看示例脚本的内容如下：

```
#查看示例脚本的内容
# cat example4.sh
#!/bin/bash

#This is a example script.
#5/14/11

#定义一个数组 ARRAY
ARRAY=(123 457 99 379 622 895 111 45 1000)
#获取数组的元素个数
LENTH=${#ARRAY[@]}
I=0
#如果变量 I 小于数组长度，则执行语句第一个 do 和最后一个 done 之间的语句块
while [ "$I" -lt "$LENTH" ]
do
    #定义一个变量 J，其值为变量 I 加 1
    J='expr $I + 1'
    #如果变量 J 的值小于数组长度，则执行下一个 do 和倒数第 2 个 done 之间的语句
    while [ "$J" -lt "$LENTH" ]
    do
       #判断第 J 个数组元素是否小于第 I 个数组元素
       if [ "${ARRAY[J]}" -lt "${ARRAY[I]}" ]
       then
       #如果第 J 个数组元素小于第 I 个数组元素
       #则将这两个元素的值交换（交换语句至 fi 语句处结束）
         TEMP=${ARRAY[J]}
         ARRAY[J]=${ARRAY[I]}
         ARRAY[I]=$TEMP
       fi
       #将变量 J 的值加 1
       J='expr $J + 1'
    done
    #将变量 I 的值加 1
    I='expr $I + 1'
done
#输出排序之后的结果
echo ${ARRAY[@]}
#清除所有变量
unset ARRAY I J TEMP LENTH
```

在上面这个示例脚本中，使用了两个递增变量 I 和 J，顺序引用、比较数组中的各元素。这个例子非常典型，初学者可以参阅 while 语句的用法仔细研究这个脚本。

示例脚本的运行结果如下：

```
#执行示例脚本，验证其功能
# ./example4.sh
45 99 111 123 379 457 622 895 1000
```

脚本的运行结果表明，脚本已正确将数组中的元素排序。

15.4.3　清除数组

不再使用数组时，应该清除数组，以回收这些数组占用的系统资源。与变量相同，清除数组也使用 unset 命令。

（1）清除数组 ARRAY 的第 1 个元素：

```
#使用 unset 命令清除数组中的第 1 个元素
# unset ARRAY[0]
```

（2）清除整个数组：

```
#使用 unset 命令清除整个数组并验证
# unset ARRAY
# echo ${ARRAY[@]}

```

从上面的命令输出中可以看到数组已经被清除。

15.5　小　　结

- 15.1 节简单介绍了什么是 Shell 脚本编程、Shell 脚本的特点等内容，以示例介绍了 Bash 脚本文件的结构、组成要素，编写时应该注意的基本原则等。在 Linux 系统中 Bash 仍然是绝对的主流，因此读者应该学习 Bash 脚本编程的相关内容。
- 15.2 节以一个示例介绍了 Tcsh 脚本的编写方法、Tcsh 与 Bash 脚本的区别等内容。Tcsh 脚本主要用于 Free BSD 等 UNIX 系统中，了解此部分的内容主要是为了一些特殊环境中的脚本移植。
- 15.3 节主要介绍了 Linux 系统中的几种变量，包括环境变量、位置变量、预定义变量和用户自定义变量等。变量在脚本编程中应用广泛，因此初学者应该掌握。
- 15.4 节讲解了 Bash 中的数组的定义、使用和清除。数组是一个非常强大的变量，在编写 Shell 脚本时经常用到，因此初学者应该掌握此部分内容。

由于 Tcsh 主要用于 Free BSD 等 UNIX 系统，因此本书中的 Shell 脚本都应用于 Bash 环境。如果需要使用 Tcsh 脚本，读者可以阅读相关文档。

第 16 章　系统脚本和登录环境

在 Linux 系统中存在许多脚本和配置文件，其中的大多数都在系统启动过程中扮演着非常重要的角色。例如全局用户配置文本负责初始化所有用户的用户环境，系统初始化脚本负责初始化系统等。作为 Linux 系统的管理员，必须了解这些脚本，因为有时不得不使用这些脚本和配置文件，例如需要手动添加随系统启动的软件、执行的命令，在启动时加载模块等。

本章将简单介绍 Linux 系统中的系统脚本及有关的配置文件，涉及的主要内容如下。

- 简单介绍 Linux 系统的启动过程。
- 介绍内核引导加载程序 grub 及涉及的配置文本和系统脚本。
- 讲解 Linux 系统初始化配置文件及涉及的系统脚本。
- 介绍全局用户文件和个人用户文件的作用及定制用户环境。

说明：许多系统脚本都是通过配置文件调用的，为方便读者理解，本章在介绍系统脚本的同时，也一并介绍相关配置文件。

16.1　系统启动过程

在学习 Linux 系统脚本之前，需要了解 Linux 系统的启动过程，以了解在启动过程中使用了哪些脚本和配置文件。本节将简单介绍 Linux 系统的引导、启动过程等。

16.1.1　Linux 系统的启动步骤

Linux 系统的启动过程是 Linux 系统的精华知识，也是系统管理中的重要知识。学好这部分知识可以了解 Linux 系统的运行机制，为定制 Linux 系统打下坚实的基础。本小节将简单 Linux 系统的启动步骤。

当主机加电时，系统启动过程就正式开始了。Linux 系统的启动过程可以简单地分为以下几步。

（1）主机加电自检：按下主机电源键之后，启动过程就开始了。系统会首先加载 BIOS（Basic Input Output System，基本输入/输出系统），检查连接到系统的设备，并枚举和初始化设备。这个过程会初始化所有连接到主机的设备，例如将光驱中的激光头复位、初始化键盘等设备。如果自检的过程中没有发现错误，系统会根据 BIOS 中的设置查找处于活动状态并能用于引导系统的设备（通常是光盘、硬盘、U 盘等），读取引导设备中的引导装载程序（有时也称引导加载程序）。

（2）引导装载程序加载内核：引导装载程序加载成功之后，系统的控制权将会交给引导装载程序（RHEL5.3 中的引导装载程序是 GRUB）。它会读取其配置文件/boot/grub/grub.conf（老版的 GRUB 配置文件是/boot/grub/menu.lst），并根据这个文件中的设置加载 Linux 内核。

（3）初始化系统环境：Linux 内核加载之后，系统的控制权将会交给内核。内核将构建最基本的内核环境，执行的工作有：调用初始化函数初始化各种设备、加载驱动和内核模块等。内核环境构建完成之后，将执行系统中的第 1 个进程 INIT。

（4）INIT 进程：INIT 进程是系统中的第 1 个进程，它是所有进程的父进程，负责管理系统中的所有进程。INIT 进程启动后会根据配置文件/etc/inittab 中的设置，进入指定的运行级别，设置网络，加载 USB 驱动模块等。

（5）加载 Login：INIT 进程最后加载的是程序/bin/login，此程序将弹出提示登录界面，要求用户输入用户名和密码登录系统。

整个启动过程中，系统会将启动过程的相关信息显示到屏幕上（也有少数系统采用图形化的方式显示启动过程的信息），管理员可以通过这些信息了解服务器的相关启动过程。

提示：少数发行版或较老的发行版使用的引导装载程序可能是 LILO，其工作原理与 GRUB 类似，感兴趣的读者可以查阅相关文档。

16.1.2　引导装载程序 GRUB

GRUB（GRUB 是 GNU GRUB 的简称）是 Linux 系统中主流的引导装载程序，许多发行版都使用它作为引导装载程序。本小节将简单介绍 GRUB 的工作过程。

说明：Linux 系统启动的第 1 步是由 BIOS 完成的，由于所有操作系统的此步骤都一样，因此本书不再介绍此过程，感兴趣的读者可以自行阅读相关文档。

引导装载程序 GRUB 加载之后，会首先读取引导分区中的配置文件/boot/grub/grub.conf，并根据这个配置文件加载内核。查看这个配置文件的内容如下：

```
#使用 cat 命令查看 GRUB 的配置文件
# cat /boot/grub/grub.conf
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/VolGroup00/LogVol00
#          initrd /initrd-version.img
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Red Hat Enterprise Linux Server (2.6.18-128.el5)
        root (hd0,0)
```

```
        kernel  /vmlinuz-2.6.18-128.el5  ro  root=/dev/VolGroup00/LogVol00
        rhgb quiet
        initrd /initrd-2.6.18-128.el5.img
```

在上面这个示例文本中，以“#”开头的行都是注释行，没有实际意义。在有效行中，设置了 GRUB 默认启动的操作系统、GRUB 引导选择界面和菜单条目（在 grub.conf 中将每个需要启动的操作系统都设置为一个菜单条目）等内容。下面将简单介绍 grub.conf 中的具体设置。

1．设置启动的操作系统

第 1、2 个有效行表示默认启动的操作系统及选择超时：

```
default=0
timeout=5
```

第 1 行表示默认启动第 1 个菜单条目，菜单条目的编号从 0 开始算起。第 2 行表示选择超时，此处的 5 表示如果用户超过 5 秒没有选择需要启动的操作系统，GRUB 将启动第 1 行中设置的菜单条目。

2．设置GRUB引导选择界面

第 3、4 行用于设置 GRUB 引导选择界面：

```
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
```

第 3 行表示 GRUB 引导选择界面使用的背景风格，此处设置的背景界面文件为第 1 块硬盘的第 1 个分区中的 grub/splash.xpm.gz。第 4 行表示显示的 GRUB 引导选择界面隐藏用户选择菜单。

3．菜单条目

菜单条目使用多行设置了一个可启动的操作系统：

```
title Red Hat Enterprise Linux Server (2.6.18-128.el5)
        root  (hd0,0)
        kernel /vmlinuz-2.6.18-128.el5 ro root=/dev/VolGroup00/LogVol00
        rhgb quiet
        initrd /initrd-2.6.18-128.el5.img
```

上面这 4 行代码就是一个菜单条目，它设置了一个可以启动的 Linux 系统。如果系统中不止一个操作系统，此处的菜单条目可能会有多个。下面将简单介绍菜单条目中的设置。

（1）菜单条目的第 1 行设置了操作系统的名称，其中 title 是新的菜单条目的起始标记。

（2）第 2 行用于指定根目录，此处设置的根目录为第 1 个硬盘的第 1 个分区（硬盘和分区编号都是从 0 算起的，其实这个分区就是引导分区）。其中的 root 表示根目录（而非 root 用户）。

（3）第 3 行用于设置 GRUB 加载的内核文件和内核参数。此处加载的是根目录中的 vmlinuz-2.6.18-128.el5（注意此时的根目录是引导分区，因此实际加载的是/boot 中的

vmlinuz-2.6.18-128.el5）。内核文件之后的是内核参数，上面的内核参数及其含义如下。

- ro root=/dev/VolGroup00/LogVol00：表示以只读方式将逻辑卷 LogVol00 挂载到根目录。
- rhgb：表示使用红帽的图形界面替代字符输出。
- quiet：表示禁止所有正常的输出。

如果读者对 rhgb 和 quiet 参数的作用不了解，不妨试试去掉这两个参数，然后重新启动系统验证效果。

（4）第 4 行用于挂载内存映像文件 initrd-2.6.18-128.el5.img，如果想要知道该文件的作用，不妨将该文件解压：

```
#在当前目录中创建一个名为 initrd 的目录并将内存映像文件复制到其中
# mkdir initrd
# cd initrd
# cp /boot/initrd-2.6.18-128.el5.img ./
#解压并恢复内存映像文件中的文件
# zcat initrd-2.6.18-128.el5.img | cpio -imd
14626 blocks
#使用 ls 命令查看恢复出来的文件列表
# ls
bin   dev   etc   init   initrd-2.6.18-128.el5.img   lib   proc   sbin   sys
sysroot
```

从上面的文件列表可以看出，该目录中有一个名为 init 的可执行脚本文件。查看这个脚本文件的内容如下：

```
#查看脚本文件 init 的内容
# cat init
#!/bin/nash

#挂载常见的文件系统
mount -t proc /proc /proc
setquiet
echo Mounting proc filesystem
echo Mounting sysfs filesystem
mount -t sysfs /sys /sys
echo Creating /dev
mount -o mode=0755 -t tmpfs /dev /dev
……
#加载 ext3 文件系统、scsi 设备等内核模块
echo "Loading ext3.ko module"
insmod /lib/ext3.ko
echo "Loading scsi mod.ko module"
insmod /lib/scsi_mod.ko
echo "Loading sd mod.ko module"
insmod /lib/sd mod.ko
echo "Loading scsi transport spi.ko module"
insmod /lib/scsi_transport_spi.ko
echo "Loading mptbase.ko module"
……
```

从上面的内容可以看出，这个 nash（nash 是一个小巧的脚本解释程序，通常在内存映

像盘中扮演交互程序的角色）脚本文件的功能是挂载文件系统，加载相应的内核模块等。由于 init 脚本文件的内容非常简单，并且许多知识已经在前面的章节中介绍过，因此此处不再赘述。

GRUB 启动 Linux 系统时，会先加载内核，然后挂载内存映像文件。最后将系统控制权交给内核，由内核执行初始化系统环境工作。

【定制内存映像文件】

学习到此，就可以定制内存映像文件了。定制内存映像文件主要是为了能加载一些特定设备的驱动模块，例如特殊的 SCSI 硬盘、特殊的阵列卡等。

（1）定制映像文件的过程非常简单，首先需要确认相应的内核模块是否已经存在目录 /lib 中。例如需要将 RAID 0 的内核模块放入内存映像文件中，先使用 locate 命令确认内核模块是否存在：

```
#使用 locate 命令查找内核模块
# locate raid0.ko
/lib/modules/2.6.18-128.el5/kernel/drivers/md/raid0.ko
```

（2）然后就可以在内存映像文件目录中，使用 mkinitrd 命令定制内存映像了：

```
#使用 mkinitrd 命令制作含有 RAID 0 模块的内存映像
# mkinitrd --with=raid0 initrd-2.6.18-128.el5.img 2.6.18-128.el5
```

需要注意的是生成的文件名的版本号（即其中的 2.6.18-128.el5，如果不知道，可以使用命令 uname –r 查看），应该与参数中的版本号一致。

当然也可以使用以下形式：

```
#使用命令置换的方式生成版本号
# mkinitrd --with=raid0 initrd-$(uname -r).img $(uname -r)
```

注意： 此处仅以 RAID 0 为例，简单讲解了如何定制内存映像文件。如果需要安装硬件驱动，建议先仔细阅读产品说明书，以获得安装驱动的帮助信息。

【GRUB 引导装载程序的工作流程】

综合本节的内容，可以得出 GRUB 启动后的简单操作流程。

（1）读取 Linux 系统的引导分区，并读取其配置文件，然后按配置文件中的置弹出引导选择界面。

（2）获得用户的选择或确定使用默认设置后，按配置文件中的指引加载内核文件，并挂载内存映像文件到内存。

（3）将系统控制权交给内核。

完成这 3 步操作后，GRUB 的工作就已经结束了。接下来内核会初始化各种硬件设备、并使用内存中的 init 脚本加载内核模块。完成后，内核会启动 Linux 系统中的第 1 个进程 INIT，并将系统的初始化任务交给 INIT。

说明： Linux 系统内核启动的第 1 个进程就是/sbin/init，启动后的进程名应该是 init（小写）。但为了区分 init 脚本，本书中遵循惯例，所有提及此进程之处都使用 INIT（大写）表示。

16.2　系统初始化过程

系统初始化过程是指 INIT 进程成功启动，一直到系统启动并弹出登录提示之间的过程。当 INIT 进程成功启动后，它会根据配置文件/etc/inittab 中的设置初始化系统，这个过程主要完成的工作有：重新挂载文件系统、运行系统需要进程和服务等。本节将简单介绍这个过程。

16.2.1　INIT 进程的配置文件

INIT 进程的配置文件是/etc/inittab，这个配置文件会引导 INIT 进程初始化系统，开启必要的进程、服务等。本小节将简单介绍 INIT 进程的配置文件。

查看 INIT 进程的配置文件内容如下：

```
#使用 cat 命令查看 INIT 进程的配置文件
# cat /etc/inittab
#
# inittab     This file describes how the INIT process should set up
#             the system in a certain run-level.
#
……
#省略部分为配置文件的注释信息
#   6 - reboot (Do NOT set initdefault to this)
#
#设置系统默认的运行级别
id:3:initdefault:

#初始化系统脚本
# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

#启动系统服务
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

#定义 Ctrl+Alt+Delete 键的作用
# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

#设置电源选项
# When our UPS tells us power has failed, assume we have a few minutes
# of power left.  Schedule a shutdown for 2 minutes from now.
```

```
# This does, of course, assume you have powerd installed and your
# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"

# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown   -c   "Power   Restored;   Shutdown
Cancelled"

#启动终端
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

与 Linux 系统中的大多数配置文件一样，所有“#”开头的行表示注释，INIT 进程会忽略这些行。对比这个文件中的有效行，读者可以轻易地发现有效行都具备一个共同的格式，即使用冒号“:”将 4 个不同的字段分隔开。查阅 man 手册可以看到每一行的格式如下：

```
id:runlevels:action:process
```

这几个字段一起定义了一个完整的进程、脚本的执行动作。这几个字段的基本含义如下（以第 1 行为例进行说明）。

- id：id 是配置行在配置文件中的标识符，最长可以由 4 个字符组成。对于大多数文本行来说，这个字段没有太大的意义，但要求每行的 id 值在该文件中是唯一的。本例中第 1 个字段是 id。
- runlevels：配置行起作用的运行级别列表。如果作用于多个运行级别，可以将其写在一起。
- action：配置应该执行的动作。通常有 initdefault、sysinit、wait、ctrlaltdel、powerfail、powerokwait 和 respawn 这几个动作。这些动作将在后面几个小节中介绍。
- process：表示配置行要执行的脚本、命令及参数等内容。

了解了以上这些基本内容之后，下一小节开始将详细讲解 INIT 进程的配置文件的作用。

16.2.2　设置系统默认运行级别

在 INIT 进程配置文件的第 1 行，设置了系统的默认运行级别：

```
id:3:initdefault:
```

在这个配置行中，动作 initdefault 用于设置系统启动时的默认运行级别，即系统启动后将自动进入的运行级别。

在上面这个示例中，系统运行级别是 3。这表示系统启动后，将默认进入完全多用户

模式。如果需要改变系统的默认运行级别，可以使用 Vi 等编辑器修改该行中的 runlevels 字段值。需要注意的是此处设置的 runlevels 字段，只能使用一个运行级别。

注意：修改系统默认的运行级别时，注意不要将该字段设置为 0 和 6，否则系统将无法正常开机。如果黑客将此字段修改为 0 和 6，通常我们将其视为拒绝服务式攻击（Denial of Service，简称 DoS）。

16.2.3　初始化系统脚本

第 2 个有效行中用于指定系统的初始化脚本：

```
# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit
```

上面这行中的 runlevels 字段为空，表示 INIT 将会忽略这一字段，并在每个运行级别都执行系统初始化脚本文件/etc/rc.d/rc.sysinit。

系统初始化脚本文件 rc.sysinit 是一个可执行文件，它是很重要的系统初始化文件，关于这个文件的内容将在 16.3 节中介绍。

16.2.4　启动系统服务

INIT 进程使用 rc.sysinit 脚本对系统进行初始化之后，就会启动系统服务。本小节将简单介绍 INIT 进程是如何启动系统服务的。

1．配置文件中的设置

在第 3 个到第 10 个有效行中，定义了系统在不同的运行级别中需要执行的脚本：

```
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
```

上面这几行依次表示在 0～6 这几个运行级别下分别要执行的脚本、命令及参数。此处的动作全是 wait，这表示 INIT 进程执行 action 字段定义的文件/etc/rc.d/rc 时，需要等待 rc 执行完毕才能执行下一步操作。

2．启动服务脚本rc

查看配置文本中指定的文件/etc/rc.d/rc 的文件类型：

```
#使用 file 命令查看/etc/rc.d/rc 的文件类型
# file /etc/rc.d/rc
/etc/rc.d/rc: Bourne-Again shell script text executable
```

可以看出这其实是一个可执行的脚本文件：

```
#查看脚本 rc 的内容
# cat /etc/rc.d/rc
#! /bin/bash
#
# rc          This file is responsible for starting/stopping
#             services when the runlevel changes.
#
# Original Author:
#             Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
#

set -m

#定义检查运行级别函数
# check a file to be a correct runlevel script
check runlevel ()
{
       # Check if the file exists at all.
       [ -x "$1" ] || return 1
       is_ignored_file "$1" && return 1
       return 0
}
……
```

从脚本文件的内容不难看出，这个文件的主要作用是接收运行级别参数，并启动对应的服务。需要启动的服务都保存在目录/etc/rc.d/rcN.d 中（通常将这些目录称为运行级别目录），此处的 N 表示运行级别。

3．运行级别目录

查看运行级别目录：

```
#使用 ls 命令查看系统中的运行级别目录
# ls /etc/rc.d/
init.d      rc0.d   rc2.d   rc4.d   rc6.d       rc.sysinit
rc          rc1.d   rc3.d   rc5.d   rc.local
```

在上面的命令输出中，有 7 个目录 rc0.d、rc1.d…rc6.d，这些目录就是运行级别目录。在这些目录中保存了相应的运行级别要启动的服务。

此处以运行级别 3 为例：

```
#使用 ls 命令查看运行级别 3 的运行级别目录中的文件列表
# ls /etc/rc.d/rc3.d/
K01dnsmasq          K85mdmpd          S10network          S44acpid
K02avahi-dnsconfd   K87multipathd     S11auditd           S50hplip
K02NetworkManager   K88wpa supplicant S12restorecond      S55sshd
……
```

从上面的命令输出中可以看出，这个目录中的所有文件都是有一个固定的格式。

- 文件名称开头的大写字母 S 和 K 分别代表 start 和 kill，表示需要运行、关闭的服务。

- 后面的两位数字表示优先级。以 S 开头的服务，系统将会按优先级从小到大的顺序启动，以 K 开头的服务则相反。
- 最后是服务名称。

掌握了上述知识之后，用户就可以快速查看运行级别 3 中服务启动的顺序了。

注意：细心的读者可能会发现运行级别目录中的文件全部都是链接文件，这些链接文件的源文件也是一个脚本文件，这些脚本文件通常称为启动脚本。关于这些启动脚本的结构和使用，将在第 19 章中详细介绍。

16.2.5　重启快捷键

第 11 个有效行中，定义了重启快捷键：

```
# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

由于 runlevels 字段为空，因此此处定义的内容将会在所有运行级别中生效。此行中的 ctrlaltdel 表示按下 Ctrl+Alt+Del 组合键，其后的 process 字段中的命令表示系统将会在 3 分钟后重新启动系统。当然许多人可能并不会使用到这个组合键，此时可以在行首加上井号“#”，让其变成注释。

16.2.6　UPS 选项

UPS（Uninterruptible Power System，不间断电源）是一种能够维持服务器稳定运行的设备。当电源出现故障时，UPS 能不间断、持续地提供一段时间的电力，以便让管理员有时间排除故障、切换后备电源或启动发电机等。关于 UPS 的应用、原理等方面的知识，感兴趣的读者可以参考相关资料和文档，此处不再赘述。

UPS 最大的作用是当市电中断时，服务器可以使用 UPS 内置的电池继续运行。有些 UPS 还可以过滤市电杂波，保护硬件设备正常运行，最大限度地防止数据丢失。

在 INIT 进程配置文件的第 12 个、第 13 个有效行，就是用于设置电力故障及电力恢复时系统采取的措施：

```
# When our UPS tells us power has failed, assume we have a few minutes
# of power left.  Schedule a shutdown for 2 minutes from now.
# This does, of course, assume you have powerd installed and your
# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"

# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown  -c  "Power  Restored;  Shutdown
Cancelled"
```

第 12 个有效行中的 powerfail 表示当接收到 UPS 发来的失去电力的信号时，系统无论处于哪个运行级别，都会向在线用户发出关闭系统的信号，并在两分钟后关闭系统。当然也可以根据实际情况修改此行中的设置。

第 13 个有效行中的 powerokwait 表示当系统接收到 UPS 发来的电力恢复信号时，如果系统正处于运行级别 1-5，将会取消关机指令，并向所有用户发送电力恢复的通知。

16.2.7　运行终端

INIT 进程启动需要的服务、设置电源选项之后，就会运行虚拟终端。本小节将简单介绍弹出终端到启动完成的配置项。

（1）第 14 个至第 19 个有效行定义了一些虚拟终端：

```
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

上面的几个配置项表示在运行级别 2～5 中，添加 6 个虚拟终端 tty1～tty6。这几个配置项中的 respawn 表示终端如果结束，就需要立即重新启动它。当然如果系统不需要那么多终端，或需要更多终端，管理员也可以修改此项设置。

（2）配置文件的最后一个有效行是运行图形终端：

```
# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

上面这个配置项表示如果系统处于运行级别 5，就运行相应的图形终端。当图形终端退出时，配置项中的 respawn 会让 INIT 进程重新启动它。

完成上述步骤后，INIT 进程就会运行/bin/login，弹出登录提示。至此系统就已经启动完成了，用户可以输入用户名和密码登录并使用系统。

16.3　系统初始化过程中使用的脚本

在 16.2 节中，讲述了 INIT 进程利用配置文件初始化系统的过程。在其中介绍了两个脚本/etc/rc.sysinit 和/etc/rc.d/rc。除此之外，还有在上一节中未介绍的/etc/rc.local 文件。需要注意的是这 3 个文件都是链接文件，其源文件保存在/etc/rc.d 目录中。本节将简单介绍这些文件的作用。

说明： 由于脚本/etc/rc.d/rc 主要用来启动服务，并且很少有管理员会用到此文件，因此本节不再介绍此脚本。

16.3.1 系统初始化脚本

从 INIT 进程的配置文件中可以看出，INIT 进程首先读取了系统默认运行级别，之后立即使用/etc/rc.d/rc.sysinit 脚本对系统进行初始化。从 INIT 进程的执行顺序上即可看出/etc/rc.d/rc.sysinit 系统初始化脚本的重要性，本小节将简单介绍系统初始化脚本的作用。

查看系统初始化脚本的内容：

```
# cat /etc/rc.d/rc.sysinit
#!/bin/bash
#
# /etc/rc.d/rc.sysinit - run once at boot time
#
# Taken in part from Miquel van Smoorenburg's bcheckrc.
#

#设置脚本
HOSTNAME=`/bin/hostname`
HOSTTYPE=`uname -m`
unamer=`uname -r`

set -m

#设置系统网络
if [ -f /etc/sysconfig/network ]; then
    . /etc/sysconfig/network
fi
……
# Let rhgb know that we're leaving rc.sysinit
if [ -x /usr/bin/rhgb-client ] && /usr/bin/rhgb-client --ping ; then
    /usr/bin/rhgb-client --sysinit
fi
```

从上面的内容可以看出，这是一个非常复杂的脚本。大多数初学者可能很难看懂这个脚本到底是如何初始化系统的。此处简单地讲解这个文件的基本功能，读者可以参照本书的后续章节阅读、学习这个文件。

系统初始化脚本 rc.sysinit 在 INIT 进程被载入之后运行，主要的功能如下。

- ❑ 获取网络环境并设置相关环境变量。
- ❑ 添加/proc、/sysfs 文件系统，检测是否存在 USB 设备，如有则尝试挂载，并载入 USB 设备驱动程序。
- ❑ 获取 SELinux 的设置状态并决定是否需要启用 SELinux。
- ❑ 从硬件时钟中读取并设置系统时钟，设置系统使用的字体、语言等。
- ❑ 设置并检测磁盘、文件系统，开启 RAID 和 LVM 支持，检测文件系统，以可读写方式重新挂载文件系统，挂载并启用本地文件系统及配额支持等。
- ❑ 将启动过程中的信息写入/var/log/dmesg 文件。

除此之外，该脚本还设置了鼠标、键盘及产生随机数设备等。系统初始化脚本最主要

的功能是为软件和系统服务创造一个良好的运行环境，许多底层功能都是在此脚本中设置完成的。

由于系统初始化脚本整个启动过程中的位置非常靠前，因此管理员通常查询这个文件，以便于修改某些设置。例如查询该文件，将载入 IPVS 模块（IP Virtual Server，IP 虚拟服务器，一种实现负载均衡的底层功能）的语句写入文件/etc/rc.modules 等。

注意：对于初学者而言，知道系统初始化文件每一行的功能是不必要的，但应该了解其在系统启动时启动的顺序和基本功能，以便于在需要时定制这个文件。

16.3.2　rc.local 脚本

系统启动的最后阶段，INIT 进程将会执行/etc/rc.local 脚本。这个脚本是通过服务的形式调用的，运行级别目录中的 S99local 就是该文件的链接。rc.local 脚本的主要作用是让用户定制系统启动时需要运行的脚本和命令。本小节将简单介绍这个文件。

查看/etc/rc.local 脚本如下：

```
#使用 cat 命令查看脚本内容
# cat /etc/rc.local
#!/bin/sh
#
# This script will be executed *after* all the other init scripts.
# You can put your own initialization stuff in here if you don't
# want to do the full Sys V style init stuff.

touch /var/lock/subsys/local
```

在脚本的最后一行，使用 touch 命令创建了一个名为 local 的空文件，这样做的目的是为了方便查阅该脚本是否已经执行过。

通常将管理员需要在系统启动时使用的命令保存在这个文件中，这样系统在启动时就会执行这些命令了。例如手动编译 MySQL 之后，可以将以下行加入该脚本：

```
/usr/local/mysql/bin/mysqld_safe --user=mysql &>/dev/null &
```

这样每次系统重新启动，MySQL 就会自动运行了。

使用 rc.local 脚本时，需要注意以下几点。

- 妥善处理命令或脚本的输出。由于 rc.local 脚本的执行位置太靠后，命令或脚本的输出可能会弄乱命令提示符的登录界面，并且出现错误也不便于查看。
- 像上面 MySQL 的例子一样，如果命令或脚本持续运行，会占用终端，应该使用&将其放入后台执行。否则用户可能会无法看到登录提示符。
- 如果要自动运行的是用户编译的程序或自己编写的脚本，建议使用绝对路径的方式表示。

本节中简单介绍了两个启动过程中运行的脚本：/etc/rc.sysinit 和/etc/rc.local，初学者需要掌握这两个脚本在系统启动过程中的执行顺序，以便按自己的需要定制。

技巧：如果系统启动之后发现系统中的某个硬件没有正常工作，可以通过运行命令 dmesg 查看系统启动过程中的初始化错误信息。

16.4 用 户 环 境

当用户在登录界面正确地输入用户名和密码后，系统就开始为用户构建一个可以使用的用户环境。用户环境包括用户使用的环境变量、快捷键设置及命令别名等。这些设置大多是通过运行全局用户配置文件/etc/profile 及用户主目录中的个人用户配置文件 profile 文件得到的。本节将简单介绍这两个文件。

16.4.1 全局用户配置文件/etc/profile

/etc/profile 是一个全局配置文件，所有用户登录都会使用该文件构建用户环境。这个文件中设置了用户的环境变量、搜索路径等信息。

```
#使用 cat 命令查看全局用户配置文件的内容
# cat /etc/profile
# /etc/profile

# System wide environment and startup programs, for login setup
# Functions and aliases go in /etc/bashrc

#设置环境搜索变量 PATH 函数
pathmunge () {
        if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
           if [ "$2" = "after" ] ; then
              PATH=$PATH:$1
           else
              PATH=$1:$PATH
           fi
        fi
}

#设置变量 EUID、UID 的值
# ksh workaround
if [ -z "$EUID" -a -x /usr/bin/id ]; then
        EUID=`id -u`
        UID=`id -ru`
fi

#如果当前登录的是 root 用户，则为 root 用户添加相应的环境变量
# Path manipulation
if [ "$EUID" = "0" ]; then
        pathmunge /sbin
        pathmunge /usr/sbin
        pathmunge /usr/local/sbin
fi

#设置资源限制
```

```
# No core files by default
ulimit -S -c 0 > /dev/null 2>&1

#为用户设置环境变量 USER、LOGNAME、MAIL
if [ -x /usr/bin/id ]; then
        USER="`id -un`"
        LOGNAME=$USER
        MAIL="/var/spool/mail/$USER"
fi

#设置主机名、命令历史的长度变量
HOSTNAME=`/bin/hostname`
HISTSIZE=1000

#环境变量 INPUTRC
#该变量主要用于加载快捷键设置
if [ -z "$INPUTRC" -a ! -f "$HOME/.inputrc" ]; then
    INPUTRC=/etc/inputrc
fi

#设置全局变量
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC

#载入别名设置
#运行 profile.d 目录中所有以.sh 结尾的脚本
for i in /etc/profile.d/*.sh ; do
    if [ -r "$i" ]; then
        . $i
    fi
done

#清除使用过的变量和函数
unset i
unset pathmunge
```

从上面的注释可以看出，这个文件的主要作用是为用户添加环境变量、设置命令别名。此处简单介绍这个文件中相应设置的作用。

- ❑ 为不同用户设置不同的搜索路径变量 PATH，以便于用户使用系统中的命令。
- ❑ 设置相关的环境变量，例如 USER、MAIL、HOSTNAME 等，以便于其他命令或脚本调用。
- ❑ 设置用户使用的别名信息，以便于用户使用别名。

除此之外，全局用户配置脚本还设置了用户使用资源的限制、快捷键等内容。用户也可以自定义这个文件，添加需要的内容，例如设置环境变量等。

16.4.2　个人用户配置文件.bash_profile

在每个用户的家目录中，还存在一个配置文件.bash_profile。每个用户登录时，系统都会运行其家目录中的.bash_profile 文件，因此这个文件多用于设置用户自己的环境。本小

节将简单介绍个人用户配置文件.bash_profile。

（1）查看个人用户配置文件的内容：

```
#使用 cat 命令查看用户主目录中的.bash_profile 文件的内容
$ cat .bash profile
# .bash profile

#判断是否存在文件~/.bashrc
#如果存在，则运行该文件
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
       . ~/.bashrc
fi

# User specific environment and startup programs

#为环境变量 PATH 添加新的搜索目录
PATH=$PATH:$HOME/bin

export PATH
```

个人用户配置文件的内容相对简单，仅实现了两个功能：其一是运行个人环境变量配置文件.bashrc；其二是为环境变量 PATH 添加新的搜索路径。

（2）在个人用户配置文件中，调用了另一个文件.bashrc：

```
#查看用户主目录中的文件.bashrc 的内容
$ cat .bashrc
# .bashrc

#判断是否存在文件/etc/bashrc
#如果存在就执行这个文件
# Source global definitions
if [ -f /etc/bashrc ]; then
       . /etc/bashrc
fi

# User specific aliases and functions
```

在个人环境变量配置文件.bashrc 中调用了全局环境变量配置文件/etc/bashrc：

```
#查看全局环境变量配置文件/etc/bashrc 的内容
$ cat /etc/bashrc
# /etc/bashrc

# System wide functions and aliases
# Environment stuff goes in /etc/profile

#设置用户的 umask 值
# By default, we want this to get set.
# Even for non-interactive, non-login shells.
```

```
if [ $UID -gt 99 ] && [ "`id -gn`" = "`id -un`" ]; then
        umask 002
else
        umask 022
fi

#设置环境变量 PS1
# are we an interactive shell?
if [ "$PS1" ]; then
    case $TERM in
        xterm*)
                if [ -e /etc/sysconfig/bash-prompt-xterm ]; then
                        PROMPT COMMAND=/etc/sysconfig/bash-prompt-xterm
……
        unset i
        unset pathmunge
fi
# vim:ts=4:sw=4
```

在全局环境变量配置文件中，设置了用户的权限掩码、环境变量 PS1 等。

16.4.3　定制用户环境

许多时候需要定制自己的工作环境，例如在环境变量 PATH 中添加目录等。这在之前的章节中已经介绍过了，此处给出定制用户环境的几点意见供初学者参考。

- 除非能肯定以后可以恢复所做的修改，否则不推荐修改全局配置文件。
- 如果确定需要修改全局配置文件，通常建议写入/etc/profile，而不是全局环境变量配置文件。因为全局环境变量配置文件是通过个人用户配置文件调用的，很可能某个用户修改了此设置。
- 如果需要做全局性设置，推荐的做法是修改个人用户配置文件，并修改/etc/skel 目录中的.bash_profile 模板文件。

此处仅仅提供了几点参考意见，任何人都可以按照自己的想法做，这正是 Linux 系统的初衷。但前提条件是，能在必要时恢复配置文件的原貌。

16.5　小　　结

- 16.1 节简单介绍了启动 Linux 系统的步骤及引导装载程序 GRUB。引导装载程序 GRUB 加载的内存映像文件，为内核提供了必要的内核驱动模块。初学者有必要了解这一点，以备不时之需。
- 16.2 节介绍了 INIT 进程初始化系统的过程。初学者需要注意 INIT 进程的配置文件中的第 1 个配置行，其运行级别字段决定了 Linux 系统启动后应该进入哪个运行级别，以便需要修改此字段进入指定的运行级别。
- 16.3 节中介绍了系统初始化过程中的两个脚本/etc/rc.sysinit 和/etc/rc.local。对于大

多数人而言，不必知道脚本中的每一句的含义，但需要了解其在系统启动过程中的顺序，以便需要时能定制。

- 16.4 节中介绍了 Linux 系统中关于用户环境的几个脚本。与系统初始化的两个脚本类似，大多数人只需要了解这几个脚本即可，无须知道每一句的含义。

本章介绍了 Linux 系统启动过程中使用的脚本及构建用户环境使用的脚本等内容。初学者了解此部分内容，不仅有助于了解系统的运行机制、提高对脚本编程的认识，还可以在必要时定制这些文件。这都是 Linux 系统管理员的基本功，因此初学者需要掌握这些脚本在系统中的作用及定制的方法。

第 17 章　函数和脚本参数

有时编写一个脚本需要反复地使用某一段具有特定功能的语句块，重复地执行这些特定功能。例如需要对几组数据取平均值时，就要反复地使用加法和除法运算。这时可以像其他编程类语言一样使用函数，将具有特定功能的语句块放入一个函数中，使用时只需要调用事先定义的函数即可。这种方法可以使程序设计者不必重复输入大量代码，也有助于提高代码的可阅读性。

本章将介绍 Shell 脚本中的函数及其使用方法、获取脚本参数等内容，涉及的知识点如下。

- 详细介绍脚本中的函数、函数的定义及使用技巧。
- 介绍如何使用并捕获传递给脚本的多个参数。

17.1　Shell 中的函数

与其他编程类语言一样，Shell 脚本也允许使用函数。函数可以将原来需要反复执行的语句块写入一次，并通过多次调用函数的方式实现语句块的功能。本节将介绍如何在脚本中使用函数。

17.1.1　在脚本中定义函数

与变量一样，在使用函数之前应该对函数进行定义。与其他编程类语言相比，由于没有数据类型的概念，因此也不必定义函数的类型。

（1）在脚本中可以使用以下方式定义函数：

```
function_name()
{
   语句 1
   语句 2
   ……
}
```

上面的语句中，function_name 为定义的函数名称，花括号内的语句 1、语句 2……等内容为函数的功能语句块，即需要重复执行的语句。

（2）有时为了便于阅读，也可以使用以下方式定义函数：

```
function function_name()
{
```

```
    语句 1
    语句 2
    ……
}
```

在上面这种方式中，使用了关键字 function 定义函数。这种方式有助于提高代码的可读性，因此建议使用这种方式定义函数。

【函数的命名规则】

虽然 Shell 中并未强制规定函数名的命名规则，但为了方便使用，通常建议命名时注意以下事项：

- 为了区别变量，建议所有函数名都由小写字母和下划线组成，并以字母开头。
- 不要使用命令作为函数名称。
- 不要在函数名中使用特殊字符。
- 函数名应该尽量体现其功能。

除此之外，尽量不要在函数名中使用数字、有特殊意义的字符（例如 true、false、return）、标点符号等。

17.1.2　在脚本中使用函数

在脚本中定义了函数之后，下一步就是在脚本中调用函数执行特定的功能，可以在脚本中使用函数名直接调用函数。

（1）下面引入一个关于函数定义和调用的简单示例脚本 example17.1.sh。在这个示例脚本中，将定义两个函数 hello 和 hi。这两个函数的功能分别为输出字符串 hello 和 hi。最后在脚本的主体中调用这两个函数，观察脚本的执行过程。

查看脚本文件的内容如下：

```
#使用 cat 命令查看示例脚本的内容
# cat example17.1.sh
#!/bin/bash

#this is a example script
#6/14/11

#定义一个名为 hello 的函数
hello()
{
  #函数的功能是使用两个 echo 命令输出字符串
  echo "Now is the function hello."
  echo "Hello!"
}

#使用关键字 function 定义了一个名为 hi 的函数
function hi()
{
  #函数功能是输出两行字符串
  echo "Now is the function hi."
  echo "Hi!"
```

```
}

#以下是脚本主体部分
#调用函数 hello
echo "Now call the function hello."
hello
#调用函数 hi
echo "Now call the function hi."
hi
```

在上面这个脚本文件中，先使用上一小节中介绍的两种方法，定义了两个函数 hello 和 hi。然后调用这两个函数，以实现相应的输出。

（2）执行脚本 example17.1.sh：

```
#执行示例脚本
# ./example17.1.sh
Now call the function hello.
Now is the function hello.
Hello!
Now call the function hi.
Now is the function hi.
Hi!
```

从示例脚本的输出结果中可以看出，脚本成功地调用了函数实现输出字符串功能。

17.1.3　向函数传递参数和返回值

许多时候需要向函数传递参数，传递参数的目的可能是为了根据参数计算某个结果，也可能是为了从参数中获取任务有关的参数等。本小节将简单介绍如何获取参数和返回函数值。

（1）在函数中获取传递的参数时，可以像脚本获取参数那样，使用位置变量$1、$2、$3 等。下面引入一个向函数传递参数的示例脚本 example17.2.sh。在这个脚本中，先定义一个函数 A，并在函数 A 中使用位置变量$1 和$2 引用传递给函数的参数，计算两个参数之和，并使用 echo 命令返回。

查看示例脚本的内容：

```
#使用 cat 命令查看示例脚本 example17.2.sh 的内容
# cat example17.2.sh
#!/bin/bash

#This is a example script.
#6/15/11

#定义一个函数 A
#function A
function A()
{
  #使用 echo 命令返回函数的计算结果
   echo 'expr $1 + $2'
```

```
}

#call the function A
#调用函数 A，并将计算结果赋值给 D
D='A 100 300'
#显示 D 的值
echo "D="$D
```

这个短小的脚本很好地说明了如何向一个函数传递参数，以及在函数中如何返回处理结果。

上面这个小脚本的运行结果如下：

```
# ./example17.2.sh
D=400
```

（2）使用 echo 命令返回函数结果的方法并不适用于所有的情况，当需要返回的值有许多时，echo 命令处理起来就比较麻烦了。这时可以借助变量，修改上面的脚本如下：

```
# cat example17.2.1.sh
#!/bin/bash

#This is a example script.
#6/15/11

#在函数中定义了一个变量 D，借助于 D 返回计算结果
#function A
function A()
{
   #将计算结果赋值给变量 D
   D='expr $1 + $2'
}

#call the function A
#调用函数 A
A 100 300
echo "D="$D
```

在上面的示例中，函数的返回值是通过变量来完成的，这种方法是借助变量的作用域在脚本中的全局性。

（3）许多时候会出现另一种情况，即需要捕获函数是否成功执行。此时许多读者可能会想起前面介绍的预定义变量$?，这个变量可以返回前一个命令的执行状态。编写脚本时也可以使用这种方式返回函数的执行状态。修改前面的示例脚本，在其中添加一个 if 判断语句。在判断语句中，根据函数返回的执行状态输出提示信息。

查看修改后的示例脚本：

```
# cat example17.2.2.sh
#!/bin/bash

#This is a example script.
#6/15/11
```

```
#定义函数 A
#function A
function A()
{
   #利用 echo 命令返回计算结果
   echo 'expr $1 + $2'
}

#call the function A
#调用函数 A，并将结果赋值给变量 D
D='A 100 300'
#通过检查预定义变量$?的值的方式检查函数是否调用成功
if [ $? = 0 ]
 then
   #如果函数 A 执行成功，则显示成功，并输出结果
   echo "Function succeeds!"
   echo "D="$D
 else
   #如果函数 A 执行失败，则输出失败提示
   echo "Function fails."
fi
```

执行脚本文件 example17.2.2.sh：

```
#执行示例脚本查看结果
# ./example17.2.2.sh
Function succeeds!
D=400
```

（4）使用 if 语句判断函数执行结果时，也可简写为：

```
if D='A 100 300'
 then
  ……
```

上面这种方式中，使用了 if 语句判断函数的执行状态，其原理与前面相同，读者可以自行修改上面的小脚本进行验证。

17.1.4　返回函数执行状态

在脚本中调用了函数之后，函数会执行函数结构体中的语句，完成后系统会根据函数中的最后一个语句的执行状态，自动返回函数的执行状态信息。但有时函数中关键的语句不是最后一句，这时就需要手动返回函数的执行状态。

要手动返回函数的执行状态，可以在函数最后使用 return 语句。return 的使用方法和取值如下。

- ❑ return：将函数最后一条命令的执行状态作为其状态信息返回。
- ❑ return 0：返回函数成功执行的信息。
- ❑ return 1：返回函数错误的信息。

使用以上命令设置函数的返回状态时，当脚本执行到 return 命令时，无论还有多少语句没有执行，脚本都会立即设置函数执行状态并返回。

17.2　在 Shell 中使用函数文件

需要编写一个较庞大的脚本时，可能会涉及许多函数、变量。这时通常建议将众多的函数、变量放入一个单独的脚本内。这样做的好处很明显，不用担心某个函数、变量是否已经被定义和使用，也不用频繁地定义、清除函数和变量。本小节将简单介绍如何使用函数文件。

17.2.1　函数文件的编写

函数文件的格式和脚本文件一样。下面引入一个由示例脚本 example17.1.sh 修改而成的函数的示例文件 function.example：

```
#使用 cat 命令查看函数文件的内容
# cat function.example
#函数文件中也可以不写下面这行 Shell 调用语句
#!/bin/bash

#function.example
#This is a function definition script.
#6/16/11

#定义函数 hello
#Defined function hello
hello()
{
  #使用 echo 命令输出字符串
  echo "Now is the function hello."
  echo "Hello! "$1"."
  return
}

#定义函数 hi
#Defined function hi
function hi()
{
  #使用 echo 命令输出字符串
  echo "Now is the function hi."
  echo "Hi! "$1"."
  return
}
```

在这个函数示例文件中，定义了两个函数 hello 和 hi。这两个函数分别输出不同的字符串，并调用传递来的参数。

17.2.2　函数文件的调用

要调用已经编写好的函数文件，可以像第 16 章中介绍的定制工作环境那样，将函数文件包含在脚本文件中，然后直接进行调用。

（1）下面引入调用函数文件的示例脚本 example17.3.sh。在这个脚本中先使用执行的方式调用函数文件，然后再直接调用函数文件中的函数。

查看示例脚本的内容如下：

```
#查看示例文件 example17.3.sh 的内容
# cat example17.3.sh
#!/bin/bash

#this is a example script
#6/15/11

#使用执行的方法调用函数文件
#function.example
. ./function.example

#调用函数文件中的函数 hello
#call function hello
echo "Now call the function hello."
#调用时传递参数
hello Jhon

#调用函数文件中的函数 hi
#call function hi
echo "Now call the function hi."
hi Alix
```

从上面这个示例文件 example17.3.sh 中可以看出，调用函数文件的格式为：

```
. ./function_file_name
```

使用这种方式调用函数文件时，函数文件的路径应该与脚本文件的路径相同，否则在调用时应该使用函数文件的绝对路径或相对路径。

执行示例脚本 example17.3.sh：

```
#执行示例脚本 example17.3.sh
# ./example17.3.sh
Now call the function hello.
Now is the function hello.
Hello! Jhon.
Now call the function hi.
Now is the function hi.
Hi! Alix.
```

从上面的输出结果中可以看出，脚本文件成功调用了函数文件中的函数，并正确输出了结果。

（2）除了上面介绍的调用方式外，函数文件也可以像变量一样在命令提示符下使用。在命令提示符中调用函数文件后，就可以直接调用函数。

在命令行提示符中调用函数文件：

```
#调用函数文件
# . ./function.example
#调用函数 hi
# hi Alix
Now is the function hi.
Hi! Alix.
#调用函数 hello
# hello Alix
Now is the function hello.
Hello! Alix.
```

用户也可以使用上面的方法，将函数文件写入用户的环境变量配置文件中，然后像使用命令一样调用自定义的函数，以实现较复杂的功能。

如果用户需要在一台主机上运行多个脚本，可以将这些脚本中的相同代码部分统一写入函数文件，然后在不同的脚本中调用，以减轻用户编写脚本时的工作量。

17.3　两个示例脚本

在前面的章节中介绍了变量和函数的使用方法。本小节将利用函数、变量和其他命令制作两个有用的脚本。

17.3.1　示例 1：检查包的依赖性

前面介绍了如何使用 RPM 包管理器管理软件包，然而许多时候一些让人摸不着头脑的提示，让许多初学者无从着手。为此可以编写一个依赖性检查脚本，让脚本自动查找软件依赖的文件所在的软件包，并输出给用户。

此处引入一个脚本文件 example17.4.sh。在这个示例脚本中，定义了一个函数 query。这个函数先计算指定目录中的 RPM 软件包数量，然后逐一查询软件包中的文件，其到找到参数传递来的文件为止。

查看示例脚本的内容：

```
#使用 cat 命令查看示例脚本的内容
# cat example15.4.sh
#!/bin/bash

#This script is used to query the package dependencies.
#6/16/11

#定义查询函数
function query()
```

```
{
  #定义查询包的行、包名称和最大行三个变量
  LINE=0
  PACKET=null
  #使用 ls 命令和 wc 命令获取参数 2 的目录中的 RPM 包数量并赋值给变量 MAX
  MAX='ls $2/*.rpm | wc -l'
  #使用一个无限循环处理包查询过程
  while true
    do

     #定义循环的内容
     #每次开始时将查询包的行增加 1
     LINE='expr $LINE + 1'
     #判断当前查询的包是否大于最大包数量，如果是则返回提示信息，并返回脚本
     if [ $MAX -lt $LINE ]
       then
        echo "Not find any package."
        #清除变量，并设置返回状态后返回
        unset LINE PACKET MAX
        return 1
     fi
     #获取变量 LINE 所指定的包名称
     PACKET='ls $2/*.rpm | sed -n ${LINE}p'
     #使用三个命令判断包内是否含有要查找的文件
     #先使用 rpm -pql 列出变量 PACKET 指向的软件包内所有的文件列表
     #同时重定向错误到系统回收池/dev/null
     #再使用命令 grep -v 去掉可能的错误和警告，并将错误重定向到系统回收池
     #最后使用 grep $1 查询第一个参数指定的文件
     #并将查找到的结果和错误都重定向到系统回收池
     rpm -pql $PACKET 2>/dev/null | grep -v "warning:" 2>/dev/null | grep
     $1 &>/dev/null
     #判断上一条命令是否成功执行，如果是则返回查找到的包名称
     if [ $? = 0 ]
       then
       #使用 echo 命令返回查找到的包名称
       echo $PACKET
       #清除使用的变量，设置返回状态
       unset LINE PACKET MAX
       return 0
     fi
    #循环结束标记
    done
}

#脚本主体部分
#检查脚本执行时是否带有一个参数
#如果没有则返回错误并退出
if [ $# != 1 ]
  then
   echo "Must have a parameter."
   echo "Usage:"$0" parameter"
   exit 1
```

```
fi

#设置 RPM 软件包所在目录变量
PACKET_DIR=/media/Server
#设置要查询的文件名称
DEPEND_FILE=$1
#设置用于获取函数返回值的变量
MESSAGE=null

#输出提示信息并开始查询
echo "Querying,please wait ..."
#调用函数并判断函数的执行状态
#函数的第 1 个参数是要查询的文件名，第 2 个参数是 RPM 软件包目录
if MESSAGE='query $DEPEND_FILE $PACKET_DIR'
  then
   #输出成功查询的提示信息，并将结果输出给用户
   echo "Query is completed."
   echo "File where the package is::"
   echo "       "$MESSAGE
   #清除使用过的变量和函数
   unset PACKET_DIR MESSAGE DEPEND_FILE query
   #设置脚本的退出状态为无错误退出
   exit 0
else
   #输出没有找到文件的相关提示信息，并设置返回状态为有错误退出
   echo "Query is completed."
   echo $MESSAGE
   #清除使用过的函数和变量
   unset PACKET_DIR MESSAGE DEPEND_FILE query
   exit 1
fi
```

在上面的脚本中，使用了非常详细的中文注释，读者只需要稍加思考即可理解这个脚本中每一句的含义。

执行上面的脚本时，首先判断脚本的参数个数是否正确，如果不正确，则提示用户并将脚本的正确使用方法输出给用户。如果参数正确，脚本将会设置安装包目录的变量 PACKET_DIR、需要查询的文件变量 DEPEND_FILE，以及用于捕获函数返回值的变量 MESSAGE。最后使用 PACKET_DIR 和 DEPEND_FILE 作为参数调用函数 query。

函数 query 被调用之后，首先使用命令 ls 和 wc 计算通过位置变量传递来的软件包目录中的 RPM 软件包数目。然后使用一个无限循环（使用了两个判断语句作为循环的出口）的形式，逐个查询所有安装包内的文件，查找位置变量传递来的文件名称，最后将结果返回给脚本。

函数返回后，脚本主体通过检测函数执行状态来判断函数是否查找到需要的软件包，然后将结果返回给用户。

执行以上脚本：

```
#不带参数检测脚本的功能
# ./example17.4.sh
Must have a parameter.
```

```
Usage:./example15.4.sh parameter
#使用脚本查询一个不存在的文件 libapbb-1.so.0 所在的软件包名称
# ./example17.4.sh libapbb-1.so.0
Querying,please wait ...
Query is completed.
Not find any package.
#使用脚本查询文件 libapr-1.so.0 所在的软件包名称
#脚本返回了正确的软件包名称
# ./example15.4.sh libapr-1.so.0
Querying, please wait...
Query is completed.
File where the package is::
       /media/Server/apr-1.2.7-11.i386.rpm
```

由于目录中的安装文件众多，脚本需要逐一检查目录中的所有安装文件，因此其执行速度可能会非常慢。

感兴趣的读者可以修改这个示例脚本，完善其功能并学习如何使用函数、变量等知识，也可以通过修改这个示例脚本逐步掌握脚本编程的要点。

17.3.2　示例 2：监控文件系统

在这个例子中将编写一个脚本监控文件系统的可用空间。当可用空间不足时，脚本会评估文件系统的写入速度和文件系统可用空间耗尽的时间，并向管理员发出告警邮件。管理员接到邮件后，就可以清理文件系统上的无用文件，以保证业务不中断。

在本例中，监控文件系统的脚本名为 monitor_disk.sh。在这个示例脚本中，定义了 3 个函数 disk_speed（用于估算文件系统的写入速度）、avai_time（用于估算文件系统耗尽的时间）、content（用于定义警告邮件的内容）。脚本主体先判断文件系统使用率是否大于 90%，如果大于 90%，将依次调用这 3 个函数估算文件系统耗尽的时间，并将结果通过邮件发给管理员。

查看示例脚本的内容如下：

```
#使用 cat 命令查看示例脚本 monitor_disk.sh 的内容
# cat monitor_disk.sh
#!/bin/bash

##This script is used to monitor disk.
#6/20/11

#定义函数 disk_speed 计算指定文件系统的写入速度
function disk speed()
{
   #通过计算 30 秒之间的文件系统使用空间之差的方法计算文件系统的写入速度
   #获取文件系统的使用空间并存入变量 A
   A='df -k | grep "$1" | awk '{print $3}''
```

```
  #暂停 30 秒
  sleep 30
  #获取 30 秒后的文件系统使用空间并存入变量 B
  B='df -k | grep "$1" | awk '{print $3}''
  #通过计算变量 A 和 B 之差再除以时间的方式获取写入速度
  D='expr $B - $A'
  SPEED='expr $D / 30'
  #返回写入速度并清除变量
  echo $SPEED
  unset A B D SPEED
}

#定义估计磁盘空间耗尽时间的函数 avai_time
function avai_time()
{
  #获取参数 1 指定的文件系统剩余空间
  FREE='df -k | grep "$1" | awk '{print $4}''
  #判断是否能计算剩余时间，如果不能就将返回时间设置为 Unknow
  #有些系统可能运行有文件系统清理脚本，因此结果可能为负，这种情况应该排除
  #如果系统没有写入，写入速度为 0，这种情况也应该排除
  #排除的情况都设置为 Unknow
  if TIME='expr $FREE / $2'
    then
      #如果剩余时间大于 60，则将时间单位转换为分钟，否则返回时间为 Unknow
      if [ $TIME -ge 60 ]
        then
         #将时间单位转换为分钟
         A='expr $TIME / 60'
         echo $A" minutes"
         unset FREE TIME
         return 0
      else
         echo "Unknow"
         unset FREE TIME
         return 0
      fi
  else
      #如果计算失败，则返回 Unknow
      echo "Unknow"
      unset FREE TIME
      return 1
```

```
  fi
}

#定义设置邮件内容函数 content
function content()
{
  #该函数的参数依次是文件系统名称、已使用空间、剩余空间、使用百分比、写入速度
  #评估的剩余时间
  #定义存放邮件内容的临时文件
  TEMP_FILE=/root/disk_monitor.tmp
  echo "Warning disk:"$1 >$TEMP_FILE
  echo "Used space:"$2 >>$TEMP_FILE
  echo "Available space:"$3 >>$TEMP_FILE
  echo "Use%:"$4"%" >>$TEMP_FILE
  echo "Write speed:"$5 >>$TEMP_FILE
  echo "Estimated time remaining:"$6 >>$TEMP_FILE
  echo "Now time:"'date +"%Y-%m-%d %H:%M"' >>$TEMP_FILE
  unset TEMP_FILE
  return 0
}

#定义要监控的文件系统
MON_DISK=/dev/md0
#获取指定的文件系统空间使用率，并使用 sed 命令删除使用率后的百分号“%”
USED='df -h | grep "$MON_DISK" | awk '{print $5}' | sed 's/%//''

#判断使用率是否大于 90%，如果是，则执行 then 后面的语句
if [ $USED -ge 90 ]
 then
   #获取已使用空间并存入变量 FREE_SPACE 中
   USED_SPACE='df -h | grep "$MON_DISK" | awk '{print $4}''
   #获取可用空间并存入变量 AVAI_SPACE 中
   AVAI_SPACE='df -h | grep "$MON_DISK" | awk '{print $3}''
   #调用 disk_speed 函数计算指定文件系统写入速度并存入变量 W_SPEED 中
   W_SPEED='disk_speed $MON_DISK'
  #判断写入速度是否等于 0
   if [ $W_SPEED != 0 ]
     then
       #如果写入速度不等于 0，则调用函数 avai_time 估算文件系统可用空间耗尽的时间
       #并将时间存入变量 S_TIME 中
       S_TIME='avai_time $MON_DISK $W_SPEED'
```

```
    else
        #如果写入速度等于 0，则将值 Unknow 存入变量 S_TIME 中
        S_TIME="Unknow"
    fi
    #调用函数 content 创建邮件内容
    content $MON_DISK $USED_SPACE $AVAI_SPACE $USED $W_SPEED $S_TIME
    #发送文件系统告警邮件
    mail -s "Disk warning" root </root/disk_monitor.tmp
    #删除使用过的临时文件
    rm -rf /root/disk_monitor.tmp
    #清除使用过的变量
    unset MON_DISK USED USED_SPACE AVAI_SPACE W_SPEED S_TIME
fi
```

上面这个脚本使用了非常详细的注释信息，读者可以很容易地读懂这个脚本中的各语句的作用。

脚本先使用变量 MON_DISK 设置要监控的文件系统，然后使用 df、grep、awk 和 sed 命令获取要监控的文件系统的使用率。使用 if 语句判断文件系统的使用率是否大于 90%，如果大于 90%，就获取已经使用的空间、可用空间，然后通过函数计算文件系统写入速度，估算可用空间耗尽的时间等。最后将文件系统的使用情况和估算结果，以调用函数的形式统一放入临时文件/root/disk_monitor.tmp 中，最后使用 mail 命令将这些内容发送给 root 用户。

执行这个脚本并查看邮件：

```
#执行脚本 monitor_disk.sh
# ./monitor_disk.sh
#使用 mail 命令查看该脚本产生的示例邮件
# mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/spool/mail/root": 2 messages 2 new
>N  1 root@FilSrv          Thu Nov 18 19:30  22/651   "Disk warning"
 N  2 root@FilSrv          Thu Nov 18 19:43  22/651   "Disk warning"
#指定查看第 1 封邮件
& 1
Message 1:
From root@FilSrv  Thu Nov 18 19:30:31 2010
Date: Thu, 18 Nov 2010 19:30:30 +0800
From: root <root@FilSrv>
To: root@FilSrv
Subject: Disk warning

Warning disk:/dev/md0
```

```
Used space:189G
Available space:14G
Use%:93%
Write speed:0
Estimated time remaining:Unknow
Now time:2010-11-18 19:29

&
```

上面这个示例脚本给出了一个文件系统监控的解决方案，读者可以改写这个脚本学习脚本编程。例如修改收邮件人的地址将其发送到自己的邮箱内；也可以将其添加为 cron 自动任务，在指定时间自动运行；还可以添加流程控制语句（循环语句），实时地监控文件系统的使用率等。

在脚本中使用函数时，建议将函数模块的功能划分得很小，以便于某个函数或功能模块出现问题时，能够很容易地排除错误。

注意：本节涉及第 18 章中的流程控制相关知识，读者可以结合第 18 章中的内容理解这个脚本。

17.4　向脚本传递参数

在第 15 章中，介绍了使用位置变量获取传递给脚本的参数的方法。但使用位置变量最多只能获取 9 个参数，当要使用的选项、参数大于 9 时，位置变量就无能为力了。本节将介绍如何引用大于 9 个的参数。

17.4.1　使用 shift 命令处理参数

在本书的前面几个章节中，介绍了使用位置变量获取传递给脚本的参数的方法，但是当参数大于 9 个时，位置变量将无法获取到 10 以后的参数。为此 Bash 引入了一个新的参数操作命令 shift。

shift 命令的用法比较特殊，它没有选项和参数，直接运行命令即可。运行 shift 命令时，位置变量就像一个可以移动的“指针”，每执行一次，位置变量指向的参数将向后移动一位。即使用 shift 命令后，位置变量$1 的值等于使用命令前的$2 的值，$2 的值等于使用命令前的$3，依此类推。随着 shift 命令的使用，位置变量$1 的值会不断变化，直到移动到最后一个参数之后。

（1）为了演示 shift 命令操作位置变量的原理和使用方法，此处引入一个示例脚本文件 example.17.5.sh。这个示例脚本会先判断传递给脚本的参数是否等于 0，如果等于 0 就调用函数 usage 输出错误信息并退出。如果参数个数不为 0，脚本将使用 while 循环和 shift 命令逐一输出所有参数。

查看示例脚本的内容：

```
#使用 cat 命令查看示例脚本的内容
# cat example17.5.sh
#!/bin/bash

#This is a example script
#6/19/11

#定义处理错误的函数 usage
function usage()
{
  #显示脚本的使用方法并返回
  echo "Error:Must have a parameter."
  echo "$0 parameter..."
  return 1
}

#判断是否传递有参数，如果没有，则调用函数 usage
if [ $# = 0 ]
  then
    usage
    exit 1
fi

#定义变量 I，用于计算变量的个数
I=1
#如果参数不为 0，则将位置变量$1 指定的参数输出
while [ $# != 0 ]
  do
    echo "\$"$I"="$1
    #使用 shift 移动位置变量“指针”
    shift
    #计数变量自动加 1
    I='expr $I + 1'
done
exit 0
```

上面这个脚本非常简单，脚本一开始就先判断脚本参数是否为 0，如果为 0，则调用函数 usage 打印出提示信息并退出。然后使用一个循环的形式，重复地移动位置变量，并将这些位置变量一一打印出来。

执行这个脚本并验证其效果如下：

```
#执行示例脚本并使用数字 1 到 10 作为参数
# ./example17.5.sh 1 2 3 4 5 6 7 8 9 10
$1=1
$2=2
$3=3
……
$8=8
$9=9
$10=10
```

从上面的输出中可以看出，重复地执行 shift 命令，使得位置变量$1 的值不断变化，并超出了 9 个位置变量的限制。

（2）为了说明 shift 命令的工作原理，改进上面的示例脚本。让脚本依次输出前 3 个位置变量的值，以观察位置变量值的移动情况：

```
#查看改进后的示例脚本内容
# cat example17.5.1.sh
#!/bin/bash

#This is a example script
#6/19/11

#定义函数 usage
function usage()
{
  echo "Error:Must have a parameter."
  echo "$0 parameter..."
  return 1
}
#判断是否传递了参数，如果没有则调用 usage
if [ $# = 0 ]
  then
    usage
    exit 1
fi
#以循环的方式移动位置并输出前 3 个位置变量
while [ $# != 0 ]
  do
    echo "\$"1"="$1"  \$"2"="$2"    \$"3"="$3
    shift
done
exit 0
```

改进后的示例脚本可以显示出前三个位置变量引用的参数变化结果。

运行脚本，其结果如下：

```
#运行示例脚本并使用字母 a 到 j 作为参数
# ./example17.5.1.sh a b c d e f g h i j
$1=a     $2=b     $3=c
$1=b     $2=c     $3=d
$1=c     $2=d     $3=e
$1=d     $2=e     $3=f
……
$1=i     $2=j     $3=
$1=j     $2=      $3=
```

从上面的脚本运行结果中可以看出，shift 命令的作用是将所有的位置变量引用的参数往后移动了一位，因此所有的位置变量引用的参数都发生了变化。

提示：使用 shift 处理参数在大多数系统管理脚本中并不多见，在与用户交互的命令式脚本中却十分常见。

17.4.2　使用 shift 命令处理选项参数

对于一个脚本编写者而言，脚本参数的含义与命令中的参数含义不同。脚本的参数应该包括类似于命令选项的选项参数，以及脚本需要操作的对象参数，它们都以参数的形式传递给脚本。因此编写脚本时，如果为脚本添加了选项参数，在处理脚本参数时，应该将选项参数和脚本参数区分开。

一个典型的处理脚本的选项参数和对象参数的格式如下：

```
while [ $# -gt 0 ]
  do
    case $1 in
      -a) #功能语句块
         #出现选项 a 时应该执行的语句
         shift
         ;;
      -b) #功能语句块
         #出现选项 b 时执行的语句
         shift
         ;;
      -c) #功能语句块
         #出现选项 c 时执行的语句
         shift
         ;;
```

```
    esac
done
```

上面这个格式中包含有 case 语句，第 18 章中将介绍这个语句的详细使用情况。

17.5 小　　结

- 17.1 节主要介绍了如何在脚本中使用函数及使用函数将脚本功能模块化的编程思路等内容。本节使用了两个实际应用的示例，讲解了函数在系统管理脚本中的应用，读者可以从这两个脚本入手了解如何使用函数。
- 17.2 节介绍了处理较庞大的脚本时使用脚本文件的情况。
- 17.3 节通过两个例子演示了 Shell 编程，复习了命令、函数的使用方法。
- 17.4 节介绍了如何向脚本传递参数，以及如何处理传递的多个参数等内容。

本章中讲解了脚本中的函数和脚本参数的应用。这些内容在编写应用脚本时经常用到，因此初学者应该掌握这些内容。

第 18 章　控制 Shell 脚本执行顺序

有时我们编写的脚本并不是顺序执行的，例如在备份文件脚本中，备份成功后脚本需要写入日志，但如果备份失败，可能就要向管理员发出邮件并将错误写入日志。这时就需要改变脚本的执行顺序，能改变脚本执行顺序的语句称为流程控制语句。

在 Shell 脚本中，提供了与 C 语言相似的流程控制语句。流程控制的工作原理是利用现有条件判断，并决定下一步应该执行的语句。例如判断是否存在某个文件，以确定写入应该是使用追加还是覆盖方式；判断某个语句是否执行成功，以确定下一步应该做什么；判断用户的输入是否符合要求，以确定是否需要让用户重新输入等。

本章将介绍 Shell 脚本编程中流程控制使用的条件，涉及的主要内容如下。

- ❑ 介绍用于改变脚本执行顺序的条件、特殊的信号，以及如何对这些条件和信号进行测试，并返回测试结果。
- ❑ 介绍 Shell 脚本中的条件测试语句 if，以及多条件测试语 case 的使用方法和应用。
- ❑ 介绍可用于循环计算的流程控制语名 for、until 和 while 的使用方法、区别和应用。
- ❑ 讲解改变流程控制的语句 break、continue 的使用方法。
- ❑ 介绍如何在脚本编程中使用流程控制语句。

18.1　条件测试和捕获信号

在 C 语言中，通常使用表达式作为改变程序执行顺序的依据。与此不同，在 Shell 脚本中，一般使用条件测试作为依据。条件测试的内容包括用户的输入、某条件命令执行完成后的返回状态、变量值、文件状态及系统发送的信号等。除此之外，Shell 脚本中的条件测试语句可以单独执行。在学习改变脚本执行顺序的语句之前，本节先简单介绍在 Shell 脚本中如何进行条件测试及捕获系统发送的信号。

18.1.1　退出状态

在 Linux 系统中，无论是命令、脚本还是程序，执行完成退出后都存在退出状态，退出状态通常保存在预定义变量$?中。大多数情况下，预定义变量都只会使用两个数字表示命令、脚本和程序的退出状态。

- ❑ 数字 0 表示命令、脚本或程序成功执行，没有发生错误。
- ❑ 数字 1 表示在执行过程中发生了某些错误，没有成功执行。

由于退出状态由程序编写者定义，因此退出状态也可能不是 0 或 1，遇到这种情况时，可能需要查阅其说明文件了解具体含义。

🔔**说明：**虽然退出状态的值可能会有许多，但只要退出状态不是数字 0，就可以认为程序、命令发生了错误。

【设置退出状态命令】

在编写较为复杂的脚本时，应该考虑错误捕捉机制，即当脚本中的语句执行出现错误时，脚本能够处理错误。一个简单的例子：如果用户使用脚本没给出必要的参数，脚本应该能够检查到这个错误，并提示用户。

在脚本中设置退出状态需要使用 exit 命令，其常见的使用形式及对应的含义如下。

- exit 0：表示返回脚本执行成功，无错误返回。这种情况有时也称为返回为真（true）。
- exit 1：表示执行失败，有错误返回。这种情况有时也称为返回为假（false）。

除了以上的 0 和 1 外，还可以使用其他一些数字，但只要返回的状态非 0，系统就认为脚本执行失败。

使用 exit 命令设置退出状态时需要注意，无论脚本执行到何处，只要遇到 exit 命令，脚本会立即设置退出状态并退出脚本。

【示例脚本】

（1）下面引入一个关于退出状态的示例脚本 example18.1.sh。在这个示例脚本中，使用 if 语句判断脚本的参数是否等于 0，如果等于 0，就调用函数 usage 并使用 exit 命令退出脚本。否则就输出参数 1 并设置退出状态为 0。

查看示例脚本的内容如下：

```
#使用 cat 命令查看示例脚本的内容
# cat example18.1.sh
#!/bin/bash

#This is a example script
#6/19/11

#定义参数错误提示函数 usage
#function usage
function usage()
{
  echo "Error:Must have a parameter."
  echo "$0 parameter..."
  return 1
}

#使用 if 语句判断脚本参数个数是否为 0
if [ $# = 0 ]
  then
    #如果参数个数为 0，则调用函数 usage 并设置有错误退出
    usage
    exit 1
fi

#如果参数个数不为 0，就输出第 1 个参数并设置无错误退出
echo $1
exit 0
```

这是一个非常简单的脚本。如果执行脚本时，用户没有输入任何参数，则调用函数 usage 输出错误提示，然后设置退出状态并退出脚本。如果有参数，则输出参数 1，然后设置无错误状态并退出。

执行这个示例脚本：

```
#执行示例脚本时不传递任何参数
# ./example18.1.sh
Error:Must have a parameter.
./example18.1.sh parameter...
#echo 命令显示脚本的退出状态为有错误退出
# echo $?
1
#执行示例脚本，并传递参数 abc
# ./example18.1.sh abc
abc
#echo 命令显示脚本无错误退出
# echo $?
0
```

从上面的脚本执行结果中可以看出退出状态命令 exit 的作用。对于初学者而言，需要注意函数返回状态命令 return 与退出状态命令 exit 的区别。

（2）对上面的示例脚本稍加修改，将函数中的 return 语句改成 exit 命令（实现从函数中直接退出），并将脚本正常退出的退出状态设置为 2：

```
#使用 cat 命令查看修改后的示例脚本 example18.1.1.sh
# cat example18.1.1.sh
#!/bin/bash

#This is a example script
#6/19/11

function usage()
{
  echo "Error:Must have a parameter."
  echo "$0 parameter..."
  #在函数中直接使用 exit 命令设置退出状态
  exit 1
}

if [ $# = 0 ]
  then
    usage
fi

echo $1
#正常退出状态设置为 2
exit 2
```

执行修改后的脚本：

```
#执行脚本时不使用任何参数
```

```
# ./example18.1.1.sh
Error:Must have a parameter.
./example18.1.1.sh parameter...
#查看脚本的退出状态
# echo $?
1
#执行脚本时传递参数 abc
# ./example18.1.1.sh abc
Abc
#查看脚本的退出状态
# echo $?
2
```

从上面的执行结果可以看出：在脚本的任何位置使用 exit 命令设置退出状态，脚本执行都会中断。

提示：在脚本编写过程中，为每一个可能的脚本退出点添加一条设置退出状态的语句是非常好的习惯。

18.1.2　文件测试

对文件的测试包括两个方面：第 1 个方面是文件基本测试，包括文件、目录是否存在、文件类型、文件长度等；第 2 个方面是文件权限测试，包括文件是否可读取、写入、执行等。本小节将简单介绍如何对文件进行测试。

1．文件基本测试

文件基本测试大多用在创建文件、目录之前，这样做的目的是让脚本拥有更好的容错性。

【文件测试命令】

文件基本测试常用的命令及其含义如下。

- d：测试目标是否存在，并且是一个目录。
- f：测试目标文件是否存在，并且是一个普通文件。
- L：测试目标是否存在，并且是一个链接文件。
- b：测试文件是否存在，并且是一个块设备文件。
- c：测试目标文件是否存在，并且是一个字符设备文件。
- e：测试指定文件或目录是否存在。

上面介绍了一些常用的文件基本测试命令，除了以上这些常用的基本测试命令以外，还有一些不常用的测试命令，感兴趣的读者可以阅读相关文档。

【文件测试命令的格式】

在进行文件测试之前，应该了解条件测试的基本格式：

```
[ -command parameter ]
```

上面的基本格式中，command 为测试命令，parameter 是需要测试的目标文件或目录。

需要注意的是，文件测试的命令、参数都要放在中括号内。

【用法示例】

（1）条件测试命令允许在命令提示符下使用。例如需要测试文件/etc/rc.local 是否为一个目录，使用如下命令：

```
#使用命令 d 测试文件
# [ -d /etc/rc.local ]
#使用 echo 命令显示退出状态
# echo $?
1
```

从命令的退出状态可以看出测试为假（也称测试失败），这表示/etc/rc.local 并不是一个目录。

注意：与 C 语言类似，在 Shell 脚本中也使用数字 0 表示真（也写作 true），非 0 数字表示假（也写作 false）。

（2）为了说明文件测试命令的用法，此处引入一个用于测试文件的示例脚本 example18.2.sh。这个示例脚本的主要功能是判断参数传递的文件的类型。脚本中的函数 test_file 先使用命令 e 测试文件是否存在，然后使用 d、c、L、b 和 f 测试文件的类型。

查看脚本的内容如下：

```
#查看示例脚本的内容
# cat example18.2.sh
#!/bin/bash

#This script is used to test the file type.
#6/20/11

#定义无参数时的返回信息函数
function usage()
{
  echo "Error:Must have a parameter."
  echo "Usage: "$0" filename"
  exit 1
}

#定义一个函数 test file，用于判断文件类型
function test file()
{
  #使用位置变量将文件名赋值给变量 FILE NAME
  FILE NAME=$1
  #使用命令 e 判断文件是否存在
  #如果不存在，则返回错误提示并退出函数
  if [ ! -e $FILE NAME ]
    then
      echo "file no find."
      return 1
  fi
  #使用多个 if 判断参数 1 传递来的文件类型
```

```
  #判断参数 1 传递来的文件名是否是一个目录，如果是则返回提示
  if [ -d $FILE NAME ]
    then
      echo $FILE NAME": Directory."
      return 0
  #判断传递来的文件是否是一个字符设备文件，如果是则返回提示
  elif [ -c $FILE NAME ]
    then
      echo $FILE NAME": Character device file."
      return 0
  #判断传递来的文件是否为一个链接文件，如果是则返回提示信息
  elif [ -L $FILE NAME ]
    then
      echo $FILE NAME": Link file."
      return 0
  #判断参数传递的文件是否为一个块设备文件，如果是则返回提示
  elif [ -b $FILE NAME ]
    then
      echo $FILE NAME": Block device file."
      return 0
  #判断参数传递的文件是否是一个普通文件，如果是则返回提示
  elif [ -f $FILE NAME ]
    then
      echo $FILE NAME": Regular file."
      return 0
  #如果文件不是以上所有的类型，则返回未知类型
  else
      echo $FILE NAME": Unknown type."
      return 1
  fi
}

#脚本主体部分
#判断是否有参数传递给脚本，如果没有就调用 usage 函数
if [ $# = 0 ]
  then
    usage
    exit 1
fi

#判断 test file 函数是否成功执行
if MESSAGE='test file $1'
  then
   #如果函数成功执行，则返回提示信息并正常退出
   echo $MESSAGE
   exit 0
else
   #如果函数没有成功执行，则返回提示信息并错误退出
   echo $MESSAGE
   exit 1
fi
```

上面这个示例脚本中，先使用 if 语句判断是否有参数传递过来，如果没有，则返回相关提示信息并退出。如果有参数传递过来，则使用函数 test_file 测试参数的文件类型。

调用函数 test_file 时，先使用命令 e 判断参数指定的文件是否存在，然后再使用其他命令判断文件的类型，并返回相应的提示信息。

运行上面的示例脚本：

```
#使用示例脚本判断/dev/sda 的文件类型
# ./example18.2.sh /dev/sda
/dev/sda: Block device file.
```

注意：链接文件也属于普通文件，因此在上面的示例脚本中，将命令 f 放置在最后一项进行测试。

2. 文件权限测试

有时可能需要在脚本中为文件写入新的内容，也可能需要读取内容或者执行某个脚本等。在对其进行操作前，通常应该检查是否具备相应的权限，以避免出现无权限拒绝访问的情况。

【文件权限测试命令】

常用的文件权限测试命令及其含义如下。

- w：判断指定的文件是否存在，并且拥有可写入权限。
- r：判断指定的文件是否存在，并且具备可读取权限。
- x：判断目标文件是否存在，并且具备可执行权限。
- u：判断目标文件是否具有 SUID 权限。

对文件权限进行测试时，需要注意的是这些测试都是针对脚本的执行用户，并非测试的是文件属主的权限。

【用法示例】

下面引入一个用于测试当前用户对参数中的文件权限的示例脚本 example18.3.sh。在这个示例脚本中，定义了一个用于生成文件权限字符串的函数 permission。这个函数会依次使用命令 e、r、w、x 检查文件的权限、生成权限字符串，并将权限字符串返回给脚本主体。

查看脚本内容：

```
#使用 cat 命令查看示例脚本的内容
# cat example18.3.sh
#!/bin/bash

#This script is used to test the file type.
#6/20/11

#定义用于返回错误信息的函数 usage
function usage()
{
  echo "Error:Must have a parameter."
  echo "Usage: "$0" filename"
```

```
  exit 1
}

#定义用于生成文件权限字符串的函数 permission
function permission()
{
  #判断传递过来的参数是否是一个文件，如果不是则返回提示
  if [ ! -e $1 ]
    then
      echo "Error:$1 file not find."
      return 1
  fi

  #判断参数 1 所指向的文件是否具有读的权限
  #并将生成的权限字符串保存在变量 PERMI 中
  if [ -r $1 ]
    then
      PERMI="r"
  else
      PERMI="-"
  fi

  #判断参数 1 所指向的文件是否具有写的权限
#并将权限标记保存在变量 PERMI 中
  if [ -w $1 ]
    then
      PERMI=$PERMI"w"
  else
      PERMI=$PERMI"-"
  fi

  #判断参数 1 所指向的文件是否具有执行的权限
  #并将权限标记保存在变量 PERMI 中
  if [ -x $1 ]
    then
      PERMI=$PERMI"x"
  else
      PERMI=$PERMI"-"
  fi

  #函数处理完成后返回生成的权限字符串
  echo $PERMI
  return 0
}

#脚本主体部分
#判断是否存在参数，如果不存在，则调用函数 usage 返回提示信息
if [ $# = 0 ]
  then
    usage
    exit 1
fi
```

```
#调用函数 permission 并判断其是否成功执行
if MESSAGE='permission $1'
  then
    #如果成功执行，则输出文件的权限
   echo $1":"$MESSAGE
   exit 0
else
   #如果没有成功执行，则输出提示
   echo $MESSAGE
   exit 1
fi
```

在上面这个示例脚本中，首先判断是否有参数传递过来，如果没有任何参数，则调用 usage 返回相关提示并退出。随后调用函数 permission 生成参数文件的权限字符串，并根据函数返回状态返回相应的值。

函数 permission 先使用了命令 e 测试参数 1 指定的文件是否存在，然后测试参数 1 传递来的文件是否具备可读、写和执行权限，最后返回文件的权限字符串。

执行上面的示例脚本：

```
#使用示例脚本判断/etc/passwd 的权限
# ./example18.3.sh /etc/passwd
/etc/passwd:rw-
```

读者可以参考这个脚本的基本形式添加 SUID 权限的测试，此处不再赘述。

技巧：初学者可以通过完善和修改示例脚本的办法学习基础知识。

18.1.3　变量测试

许多时候需要对变量进行测试，对变量的测试内容是测试变量是否已经被定义（被定义的标准是变量已经赋值）。本小节将简单介绍如何测试变量。

测试变量是否被定义需要使用命令 z，这个命令的格式与上一小节中的文件测试命令相同。但与前面介绍的测试命令相反的是，对于没有被定义的变量，将会返回数字 0，已经定义的函数将会返回数字 1。

在命令提示符下测试变量：

```
#使用命令 z 测试变量 NAME 是否被定义
# [ -z $NAME ]
#从 echo 命令的显示可以看出变量未被定义
# echo $?
0
#为变量 NAME 赋值并重新测试变量
# NAME=Jhon
# [ -z $NAME ]
#echo 命令显示变量已经被定义
# echo $?
1
```

在上面的示例命令中，没有设置任何值的变量，测试返回数字 0；而设置了变量的值后，测试返回数字 1。

18.1.4　字符串和数值测试

许多时候都需要对字符串内容和数值进行测试，这些测试可能是为了验证用户的某个输入，也可能是为了测试某个变量值是否在指定范围等。本小节将简单介绍在脚本中如何进行字符串和数值测试。

小知识： 在其他编程类语言中也存在字符串和数值测试，这些编程类语言大多使用“>”、“<”表示大于和小于。甚至有些编程类语言中还允许使用远大于“>>”和远小于“<<”比较操作符。与这些编程类语言不同的是，由于在 Shell 中操作符号“<”和“>”用于重定义输入和输出，因此不能使用这两个符号进行测试。

1．字符串测试

在 Shell 中用于对字符串进行测试时，通常都是对字符串进行匹配操作（即字符串是否等于某一个特殊的值）。例如需要判断用户输入的是否为 yes 或 no，就需要使用字符串测试。

【字符串测试的操作符】

字符串测试的操作符及含义如下。

- =：判断两个字符串是否相等，如果相等，则返回为真（即数字 0）。
- !=：判断两个字符串是否不相等，如果不相等，则返回为真。
- n：测试字符串是否为非空。

使用字符测试时，最为常见的例子是判断某个变量是否为空，或者某个变量是否为某个特定值。

【字符串测试的格式】

使用字符串测试时的格式如下：

```
[ parameter1 operator parameter2 ]
```

在上面的格式中，parameter1 和 parameter2 分别表示字符串 1、字符串 2，operator 表示操作符。

【用法示例】

（1）字符串测试可以在命令行提示符中进行。例如测试两个字符串是否相等：

```
#测试字符串 abc 是否等于 ABC
# [ "abc" = "ABC" ]
#echo 命令显示结果为不相等
# echo $?
1
```

（2）许多时候都希望能够判断某个变量是否为空：

```
#在字符串测试中使用变量
# [ "$NAME" = "" ]
```

```
# echo $?
1
```

从上面返回的结果可以看出，变量 NAME 并非为空。

（3）测试某个变量是否等于一个字符串：

```
#测试变量 NAME 中的值是否等于 Jhon
# [ "$NAME" = "Jhon" ]
# echo $?
0
```

（4）使用命令 n 测试字符串是否为空：

```
#使用命令 n 测试字符串是否为空
# [ -n "$NAME" ]
#测试结果显示为空
# echo $?
1
#为变量 NAME 赋值，然后使用命令 n 测试是否为空
# NAME="Jhon"
# [ -n "$NAME" ]
# echo $?
0
```

为了避免系统误解，无论使用何种操作符、命令对字符串进行测试，都建议将变量或字符串放入引号“""”内。

2．数值测试

在脚本编写时，许多时候都会使用到数值测试（即比较两个数的大小）。最为常用的例子：使用数值测试检查脚本的参数个数，使用数值测试为循环设置出口等。

【数值测试命令】

数值测试使用的命令及其含义如下。

- ❑ eq：如果两个数相等，则返回为真。
- ❑ ne：如果两个数不相等，则返回为真。
- ❑ lt：如果第 1 个数小于第 2 个数，返回为真。
- ❑ le：如果第 1 个数小于等于第 2 个数，则返回为真。
- ❑ gt：如果第 1 个数大于第 2 个数，则返回为真。
- ❑ ge：如果第 1 个数大于等于第 2 个数，则返回为真。

使用上面的命令进行数值测试时，如果要判断两个数是否相等，也可以使用字符串测试中的“=”和“!=”进行测试。

注意：使用数值测试命令时，其格式与字符串测试的格式基本相同。

【用法示例】

（1）测试数字 300 是否小于 200：

```
#使用命令 lt 测试数字 300 是否小于 200
#从命令的返回状态可以看出测试为假
```

```
# [ 300 -lt 200 ]
# echo $?
1
```

上面的返回结果表明数字 300 不小于 200。

（2）进行数值测试时，也可以将两个数放入引号内：

```
#在数值测试中使用引号
# [ "300" -lt "500" ]
# echo $?
0
```

无论使用哪种形式进行测试，命令都会先将两边的字符转换为数值之后，再进行数值测试。

（3）对两个变量进行测试：

```
#设置两个变量 A、B 的值
# A=14
# B=53
#对变量进行数值测试
# [ $A -gt $B ]
# echo $?
1
```

（4）测试某个变量是否大于等于某个数字：

```
# FLAG=134
#在数值测试中使用变量和数字
# [ $FLAG -ge "133" ]
# echo $?
0
```

从上面的输出可以看出，测试成功并返回为真。

18.1.5　逻辑操作符

许多时候都希望能够在测试时加入逻辑操作符，进行比较复杂的判断。例如文件同时满足可写、可读等，变量的值在第 1 个数和第 2 个数之间等。本小节将介绍如何使用逻辑操作符实现复杂的测试。

【常用的逻辑操作符】

在 Shell 条件测试中常用的逻辑操作符及其含义如下。

- a：逻辑与，操作符两边都为真时，结果为真，否则为假。
- o：逻辑或，操作符两边至少有一边为真时，结果为真，否则为假。
- !：逻辑非，条件为真时，结果为假，条件为假时，结果为真。

上面的逻辑操作符中，除了逻辑非以外，逻辑与和逻辑或通常都需要使用两个条件测试。

【用法示例】

（1）测试文件/etc/passwd 是否可读、写：

```
#使用逻辑操作符 a 判断文件/etc/passwd 是否可读、写
# [ -r /etc/passwd -a -w /etc/passwd ]
# echo $?
0
```

（2）测试文件可读或可执行：

```
#使用逻辑操作符 o 判断文件/etc/passwd 是否可读，或可执行
# [ -r /etc/passwd -o -x /etc/passwd ]
#由于文件/etc/passwd 可读，因此返回为真
# echo $?
0
```

技巧： 有时也可以使用其他方法测试，例如使用 ifconfig 命令判断 IP 地址是否合法等。

18.1.6　捕获系统信号

在编写脚本时，可能会担心由于脚本的使用者发出了一个终止脚本运行的信号，Shell 脚本就终止运行，从而无法完成正在处理的工作。如果脚本正在处理一些无关紧要的事，例如在文件系统上查找文件，脚本终止运行可能不会有任何损失。但如果正在执行一项非常重要的工作，例如备份文件系统中的数据，程序终止就可能会带来损失。因此在重要的脚本中，应该捕获系统发出的终止信号并做出相应的动作，以免脚本终止带来不必要的损失。本小节将简单介绍如何捕获系统信号。

【可以捕获的信号】

在学习捕获系统信号之前，应该了解哪些信号可以被捕获。如果是信号 9，可能根本就无法捕获到任何信号，因为系统会将程序强制结束并回收脚本使用的资源。

通常情况下，可以捕获的信号编号、名称及其含义如下。

- 1（SIGHUP）：来自控制终端的挂起或来自控制进程的死亡信号。
- 2（SIGINT）：来自键盘的中断信号。
- 3（SIGQUIT）：来自键盘的退出信号。
- 15（SIGTERM）：终止信号。

如果不确定脚本会捕获到哪种信号，建议在重要的脚本中，捕获所有可以捕获的信号，以免带来不必要的损失。

【捕获信号后的措施】

在捕获到系统信号之后，脚本应该立即执行采取措施。通常可以采取以下 3 种措施。

- 不采取任何措施，由系统处理这些信号。这会导致脚本结束执行、暂停等后果，这个办法通常用于不重要的脚本中。
- 捕获并忽略信号，即捕获到信号后不采取任何操作。
- 捕获信号并采取操作，这通常用于比较重要的脚本中。通常操作是提示用户是否需要中断脚本的执行，也可以向用户发出脚本正在执行不能退出的消息等。

在了解了以上内容之后，就可以视脚本的重要程度为其设置相应的措施了。

【捕获信号的格式】

捕获系统信号使用命令 trap，其基本格式如下：

```
trap "command" signals
```

在上面的格式中，command 表示捕获到信号之后需要执行的命令或函数，如果为空（使用""表示）则表示忽略信号，否则表示执行信号。signals 表示要捕获的信号列表，可以使用数字表示，也可以使用信号名称表示。如果要捕获多个信号，则使用空格作为信号间的分隔符。

【用法示例】

此处引入一个捕获信号的示例脚本 ex_trap.sh。在这个脚本中，将要捕获的是用户从键盘上发送的结束信号（按键为 Ctrl+C，对应的信号为 2），捕获到信号后脚本将调用函数 trap，输出提示信息并退出。

查看脚本的内容：

```
#使用 cat 命令查看示例脚本的内容
# cat ex_trap.sh
#!/bin/bash
#this is a example script

#如果捕获到信号 2，则执行函数 trap
trap "trap" 2

#定义捕获到信号后的处理函数 trap
function trap()
{
   echo "You press the Ctrl+C."
   echo "Exiting,please wait..."
   exit 1
}

sleep 60
```

这是一个用于捕获 Ctrl+C 退出快键的脚本，运行其结果如下：

```
#执行示例脚本
# ./ex_trap.sh
#此时按下 Ctrl+C
You press the Ctrl+C.
Exiting,please wait...
```

按下快捷键 Ctrl+C 时，可以看到脚本捕获到了用户从键盘发出的中断信号，并调用函数 trap，显示提示信息然后退出。

此处简单讲解了信号捕获的方法和作用，感兴趣的读者可以阅读相关资料，掌握更多关于信号捕获的应用。

18.2　条件判断语句 if

在编写脚本时，经常需要判断一些情况，例如判断某个值的大小、判断执行脚本的用户等。像这些任务都可以使用条件判断语句 if 来完成。本节将简单介绍 if 语句在脚本编程中的用法。

18.2.1　简单 if 语句的使用

Shell 脚本编程中的 if 语句结构和使用都非常灵活，其结构可以被拆分为好几种常见的形式。本小节将简单介绍最简单的 if 语句的使用方法。

【简单 if 语句的格式】

最简单、常见的 if 语句格式如下：

```
if 条件表达式
  then
    语句 1
    语句 2
    ……
fi
```

在上面的 if 语句格式中，条件表达式应该是上一节中介绍的条件测试。这个简单 if 语句的执行流程如图 18.1 所示。

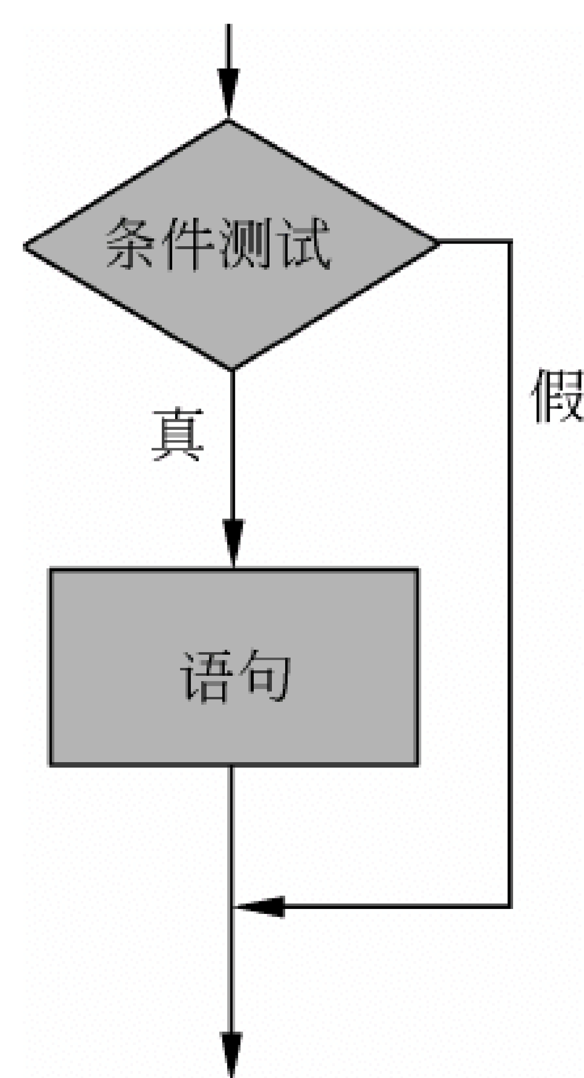


图 18.1　简单 if 语句的执行流程

从图 18.1 中可以看出，当条件测试返回为真时，脚本将会执行 then 后面的语句。最后一行的 fi 是 if 语句的结束标记，执行中遇到 fi 语句，就表示 if 语句已经执行完成。语句 then 和 fi 之间的是由多个语句组成的特定的功能模块，例如调用特殊的函数、跳出循环等。

【用法示例】

使用 if 语句简单结构的例子：判断是否存在脚本参数以便于提示用户输入参数、是否具备某种退出的条件等。

（1）下面是一个使用 if 语句判断脚本参数个数的示例脚本，这在之前的脚本中已经介绍过。

```
#使用 cat 命令查看示例脚本 example18.4.sh 的内容
# cat example18.4.sh
#!/bin/bash

#This is a example script
#6/22/11
```

```
#定义参数错误提示函数 usage
function usage()
{
   echo "Error:Must have a parameter."
   echo "Usage:$0 parameter..."
}

#使用$#判断参数个数是否为 0
if [ $# = 0 ]
  then
    #如果参数个数为 0，就调用函数 usage 输出提示信息，并退出
    usage
    exit 1
fi

#参数个数不为 0，就输出所有参数
echo $#
exit 0
```

上面这个脚本在前面的章节中已经介绍过，这是一个使用 if 语句判断传递给脚本参数个数的示例。脚本先使用 if 语句判断传递来的参数个数是否为 0，如果为 0，就调用函数 usage 输出提示信息，如果不等于 0，就将参数个数输出。

执行上面的脚本：

```
#执行脚本并且不传递参数时，脚本输出了提示
# ./example18.4.sh
Error:Must have a parameter.
Usage:./example18.4.sh parameter...
#为脚本提供一个参数时，脚本输出了参数的个数
# ./example18.4.sh abc
1
```

（2）有时需要控制脚本的使用者，例如包含管理命令的脚本只能以 root 用户的身份来执行。这时就需要在脚本中使用 if 语句判断当前用户是否为 root，如果不是就输出提示，并退出脚本。判断依据通常是判断环境变量 UID 的值或命令 whoami 的输出（该命令将输出当前用户名）。

查看脚本的内容如下：

```
#使用 cat 命令查看示例脚本的内容
# cat example18.5.sh
#!/bin/bash

#This is a example script
#6/22/11

#使用 if 语句判断用户的 UID 是否不等于 0
```

```
if [ $UID != "0" ]
  then
    #如果 UID 不等 0，就提示用户需要以 root 用户的身份运行此脚本
    echo "Error:Must use the root user to run the script !"
    exit 1
fi

#如果是 root 用户，就打印提示
#Add script content
echo "You are using root user..."
```

在上面这个示例脚本中，使用 if 语句判断当前用户的 UID 号是否为 0，如果不为 0，则返回相关提示信息并退出。

运行脚本如下：

```
#以 root 身份运行示例脚本
# ./example18.5.sh
You are using root user...
#以普通用户身份运行脚本时，出现了错误
$ ./example18.5.sh
Error:Must use the root user to run the script !
```

从上面的输出中可以看出，脚本已经实现了判断 root 用户的功能。

技巧：如果 if 语句实现的功能简单，也可以使用逻辑运算符&&和||替代。

注意：在 Shell 脚本中并没有规定 if 语句必须使用缩进，因此也可以将 if 语句写在同一行。但为了提高脚本的可阅读性，建议使用缩进等方式。

18.2.2　if else 语句的使用

有时需要在 if 语句条件测试成功时执行一个功能模块，条件测试不成功时，也要执行功能模块。这时就可以使用 if else 语句，它允许条件测试成功时执行一组功能性语句或模块，条件测试不成功时也执行一组功能性语句或模块。

【if else 语句的格式】

在 shell 中 if else 语句的基本格式如下：

```
if 条件测试
  then
     语句块 1
esle
     语句块 2
fi
```

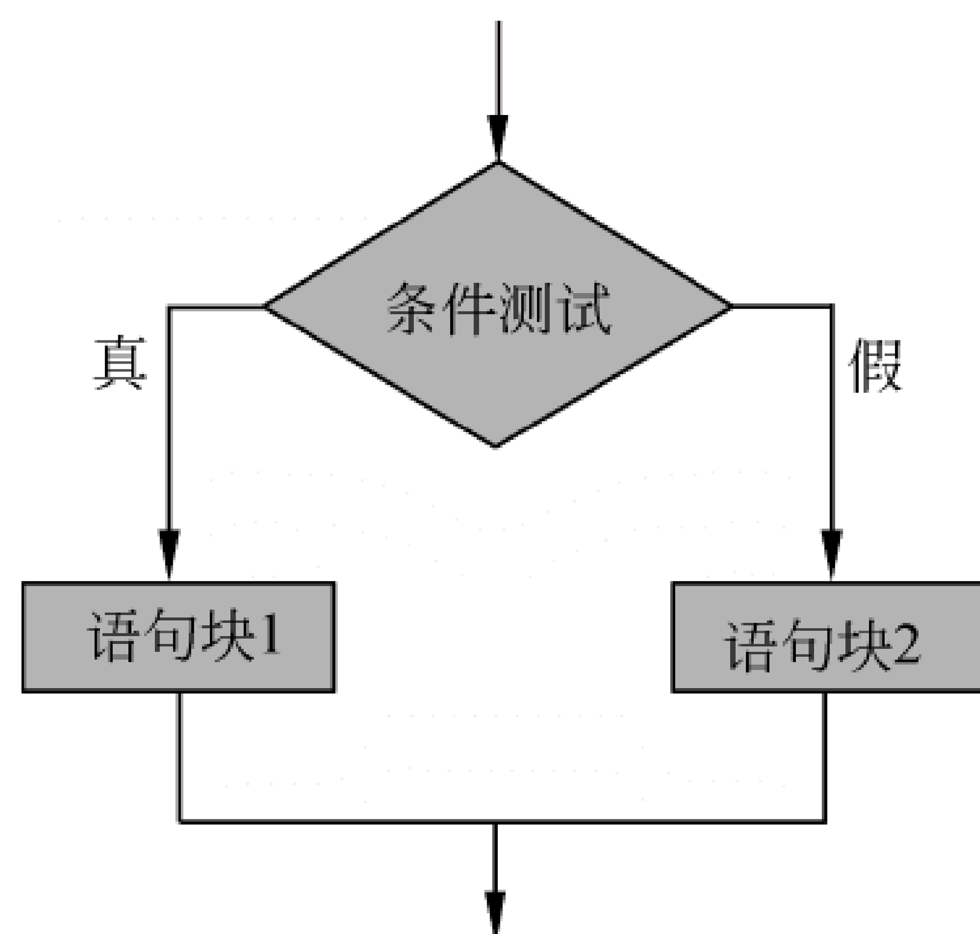


图 18.2　if else 语句的执行流程

if else 语句的执行流程如图 18.2 所示。

从图 18.2 中可以看出，如果条件测试为真，系统会执行 then 语句到 else 之间的语句块 1，否则会执行 else 语句后面的语句块 2。

【用法示例】

if else 语句在编程中经常用来处理程序中的两面性问题，例如变量大小、文件是否存在、文件是否可写等。

（1）此处引入第 15 章中介绍的使用 if else 语句猜数字的游戏：

```
#使用 cat 命令查看示例脚本的内容
# cat example5.sh
#!/bin/bash

#this is a example
#5/14/11

#产生一个 10 以内的随机数
#使用取余的方法获取 10 以内的随机数
RAN='expr $RANDOM % 10'
#由于产生的随机数可能是 0，因此需要加 1
RAN='expr $RAN + 1'
echo "This is a guessing game."
echo "Number range:1-10"
#使用一个循环判断用户的输入
while true
  do
   #提示用户输入数字
   echo -n "Please enter the number you think:"
   read NUM
   #判断用户是否在 1~10 以内，如果不在 1~10 范围以内，则提醒用户重新输入
   if [ $NUM -lt 1 ] || [ 10 -lt $NUM ]
    then
     echo "You enter the number out of range."
     continue
   fi
   #判断用户输入与产生的随机数是否相等
   #如果用户输入与随机数相等，则输出提示信息，清除相关变量并退出
   if [ $NUM = $RAN ]
    then
     echo "You guessed it."
     unset NUM RAN
     exit 0
   fi
   #判断用户输入是否小于随机数，如果小于随机数，则产生相关提示
   if [ $NUM -lt $RAN ]
    then
     echo "You enter the number is too small."
   else
     echo "You enter the number is too large."
   fi
done
```

上面的脚本最后使用 if else 语句判断用户输入的数字是否小于随机数，如果小于随机数，则输出提示用户输入数字太小的信息。如果测试不成立，就输出用户输入的数字太大的提示信息。

（2）下面是一个使用 if else 语句处理文件写入权限的示例脚本。在这个示例脚本中，先使用 if 语句判断文件 test 是否可写，如果可写，则将临时文件中的内容写入文件中，然后删除临时文件。否则就认为文件不可写，输出错误信息后退出。

查看示例脚本的内容如下：

```
#使用 cat 命令查看示例脚本的内容
# cat example18.6.sh
#!/bin/bash

#this is a example script
#6/22/11

#使用 if 语句判断./test 文件是否拥有可写权限
if [ -w ./test ]
  then
    #如果有可写权限，就将文件./test.tmp 中的内容写入./test 文件中
    cat ./test.tmp >./test
    #输入完成后删除临时文件 test.tmp 并退出
    rm -rf test.tmp
    exit 0
else
    #如果文件不可写，就输出错误提示
    echo "Error:Target file is not writable." >2
    exit 1
fi
```

（3）有时 if else 也经常用于判断脚本中某些关键语句是否能正确执行。例如下面这个检测主机连通性的例子，如果 ping 命令的退出状态为真，就输出能够连通的信息，否则就输出不能通信的错误信息。

查看示例脚本的内容如下：

```
#使用 cat 命令查看示例脚本的内容
# cat example18.7.sh
#!/bin/bash

#This is a example script.
#6/22/11

#使用 ping 命令发送 3 个数据包测试与主机间的连通性
if 'ping 192.168.104.122 -c 3 &>/dev/null'
  then
    #如果能连通，就输出提示信息，然后退出
    echo "Host can communicate with the target."
    exit 0
else
    #否则就认为与主机不可通信
    echo "Can not communicate with the target host."
    exit 1
fi
```

执行上面的示例脚本如下：

```
#使用示例脚本测试连通性
# ./example18.7.sh
Can not communicate with the target host.
```

18.2.3　if elif 语句的使用

在使用 Shell 编写脚本的过程中，可能会遇到多条件检测的问题。例如要判断某个变量的取值处于哪个范围，以执行不同的操作等，这时就可以使用 if elif 执行多个条件检测。

【if elif 语句的基本格式】

多条件检测语句 if elif 的基本格式如下：

```
if 条件测试 1
  then
    语句块 1
elif 条件测试 2
  then
    语句块 2
elif 条件测试 3
  then
    语句块 3
elif 条件测试 4
    语句块 4
……
else
    语句块 5
fi
```

在上面的基本格式中，if elif 语句有许多个条件测试语句。其执行流程如图 18.3 所示。

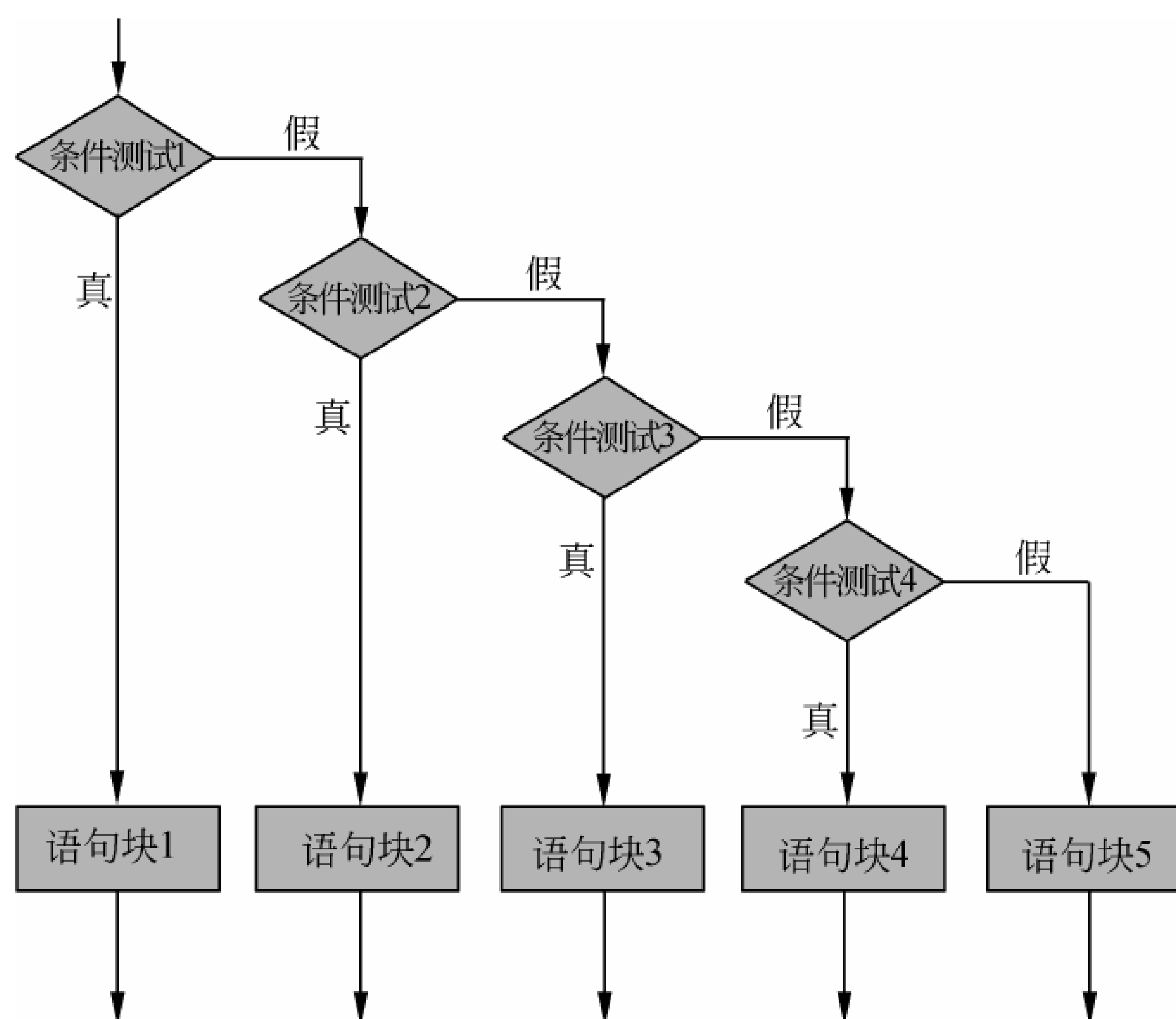


图 18.3　if elif 语句执行流程

从图 18.3 中可以看出，执行时从上至下先判断条件测试 1，如果满足，就执行 then 后

面的语句块 1。如果不满足，则继续判断条件测试 2，如果条件测试 2 满足，就执行条件测试 2 的 then 语句后面的语句块 2。然后判断条件测试 3，依此类推，直到将所有的条件测试判断完成。如果所有的条件测试都测试完成，仍然没有一个条件测试能够满足，将会执行 else 语句后面的语句块 5。

【用法示例】

（1）为了说明 if elif 语句的用法，此处引入本章前面介绍的示例脚本 example18.2.sh：

```
# cat example18.2.sh
#!/bin/bash
……
#省略的部分请参考前几节中的内容

#定义一个函数 test_file，用于判断文件类型
function test file()
{
  #使用位置变量将文件名赋值给变量 FILE_NAME
  FILE NAME=$1
  #使用命令 e 判断文件是否存在
  #如果不存在，则返回错误提示并退出函数
  if [ ! -e $FILE NAME ]
    then
      echo "file no find."
      return 1
  fi
  #使用多个 if 判断参数 1 传递来的文件类型
  #判断参数 1 传递来的文件名是否是一个目录，如果是则返回提示
  if [ -d $FILE NAME ]
    then
      echo $FILE NAME": Directory."
      return 0
  #判断传递来的文件是否是一个字符设备文件，如果是则返回提示
  elif [ -c $FILE NAME ]
    then
      echo $FILE NAME": Character device file."
      return 0
  #判断传递来的文件是否为一个链接文件，如果是则返回提示
  elif [ -L $FILE NAME ]
    then
      echo $FILE NAME": Link file."
      return 0
  #判断参数传递的文件是否为一个块设备文件，如果是则返回提示
  elif [ -b $FILE NAME ]
    then
      echo $FILE NAME": Block device file."
      return 0
  #判断参数传递的文件是否是一个普通文件，如果是则返回提示
  elif [ -f $FILE NAME ]
    then
      echo $FILE NAME": Regular file."
      return 0
  #如果文件不是以上所有的类型，则返回未知类型
```

```
  else
      echo $FILE NAME": Unknown type."
      return 1
  fi
}

#省略部分请参考前几节中的内容
……
```

在上面这个示例脚本中，函数 test_file 使用了一个 if elif 语句，依次判断通过参数传递来的文件属于哪种类型，并在相应的类型后设置提示信息。

（2）有时需要在脚本中使用文件目录，但可能并不知道这些目录是否能被使用，例如是否能被写入等。下面引入一个查找可用临时目录的脚本。

在这个示例脚本的 tmp_dir 函数中，先判断系统临时文件目录/tmp 是否可以写入。如果可以写入，就返回这个临时文件目录。不能写入则继续判断目录/var/tmp 是否可以写入，可以写入就返回该目录。如果这两个临时文件目录都无法写入文件，就在用户主目录中新建一个名为.tmp 的隐藏目录并返回。最后脚本主体根据 tmp_dir 函数返回的临时文件目录，将当前目录中的所有临时文件都移动到返回的临时文件目录中。

查看脚本的内容如下：

```
#使用 cat 命令查看示例脚本的内容
# cat example18.8.sh
#!/bin/bash

#This is a example script.
#6/23/11

#定义搜寻临时文件目录函数 tmp_dir
function tmp dir()
{
   #判断/tmp 目录是否可写
   if [ -w /tmp ]
     then
       #如果/tmp 目录可写就返回目录名
       echo "/tmp"
       return 0
   #如果/tmp 目录不可写，就判断/var/tmp 目录是否可写
   elif [ -w /var/tmp ]
     then
       #如果/var/tmp 目录可写就返回目录名
       echo "/var/tmp"
       return 0
   else
       #如果以上两个目录都不可写，就在家目录中创建一个临时文件目录~/.tmp
       #并使用 echo 命令返回创建的临时文件目录
       mkdir -p ~/.tmp &>/dev/null
       echo "~/.tmp"
       return 0
   fi
}
```

```
#调用函数并将临时文件目录存放在变量 TMP_DIR 中
#然后移动当前目录中的所有临时文件
TMP_DIR='tmp_dir'
echo "TMP_DIR="$TMP_DIR
mv *.tmp $TMP_DIR &>/dev/null
exit 0
```

运行上面这个示例脚本：

```
#执行上面的示例脚本，判断可以写入的临时文件
# ./example18.8.sh
TMP_DIR=/tmp
```

从上面的脚本输出中可以看出，脚本找到了可以写入的临时文件目录。

18.2.4　多 if 语句嵌套

许多时候会遇到使用 if 语句判断之后，在功能性语句中又需要使用 if 语句判断的情况。这时可以使用多个 if 语句嵌套的方法，即在一个 if 语句中再嵌入多个 if 语句。

在本小节之前的脚本示例中，已经存在嵌入多个 if 语句的例子。此处再引入一个示例脚本 example18.9.sh。在这个示例脚本中，先判断是否有参数传递，如果没有，则输出提示信息并设置退出状态后退出。有参数传递时，就先判断参数 1 是否为一个目录，如果是则输出提示信息，在经过用户的确认之后将目录删除。如果参数 1 不是一个目录，则询问用户是否需要删除参数传递来的文件，在用户确认之后删除参数指定的文件。

查看示例脚本的内容如下：

```
#使用 cat 命令查看示例脚本的内容
# cat example18.9.sh
#!/bin/bash

#This is a example script.
#6/23/11

#定义参数不正确的提示信息函数 usage
function usage()
{
  echo "Error:must have a parameter."
  echo "Usage:$0 filename"
}

#如果参数个数等于 0，就调用 usage 函数输出提示信息
if [ $# = 0 ]
  then
    usage
    exit 1
fi

#判断参数 1 是否为一个目录
if [ -d $1 ]
```

```
 then
   #如果参数 1 是一个目录，就提示用户是否需要删除目录，并读取用户的输入
   echo -n "Sure you want to delete the directory?[y|n]"
   read ANS
   #使用 if 语句判断用户输入的是否为 y
   if [ $ANS = "y" ]
     then
       #如果用户输入为 y，就删除参数 1 指定的目录
       rm -rf $1 &>/dev/null
   else
       #如果用户输入的不是 y，就退出脚本
       exit 1
   fi
else
   #如果参数 1 不是目录，就提示用户是否删除文件，并接受用户的输入
   echo -n "Sure you want to delete the file?[y|n]"
   read ANS
   #如果用户输入的是 y，就删除参数 1 指定的文件，否则就退出
   if [ $ANS = "y" ]
     then
       rm -rf $1 &>/dev/null
   else
       exit 1
   fi
fi
```

上面的示例脚本中，在一个 if 语句中嵌入了两个不同的 if 语句。如果需要，还可以嵌入更多 if 语句。

注意：虽然在 Shell 脚本中嵌入的 if 语句的个数并没有限制，但通常都不建议嵌入过多的 if 语句。因为这会降低脚本可阅读性，并大大增加维护时的难度。如果确实需要嵌入多个 if 语句，可以将部分 if 语句写入函数中，以提高脚本的可阅读性。

18.3　多条件判断语句 case

在编写脚本的过程中，可能需要判断用户通过参数传递过来的选项参数。选项参数可能用于改变脚本的运行模式，也可能是功能性的选择等。

一个最典型的例子：制作一个带菜单的脚本，用户通过提示输入一个命令，以执行菜单中对应的功能。如果使用 if 语句来实现这个功能，可能会使用许多个 if elif 来判断可能的用户输入，这是非常不方便的，因为需要使用许多个 if elif 语句来判断各种情况。这时可以使用 case 语句来实现这个简单的菜单功能，本节将简单介绍 case 语句的使用方法。

18.3.1　多条件判断语句 case 的基本格式

如果一个变量可能包括的值有许多，使用 if 语句处理变量时就会非常麻烦。这时可以

使用 case 语句。本小节将简单介绍 case 语句的基本格式。

case 语句的基本格式如下：

```
case 变量 in
  模式 1)
      语句块 1
      ;;
  模式 2)
      语句块 2
      ;;
  ……
      ;;
esac
```

case 语句的执行流程如图 18.4 所示。

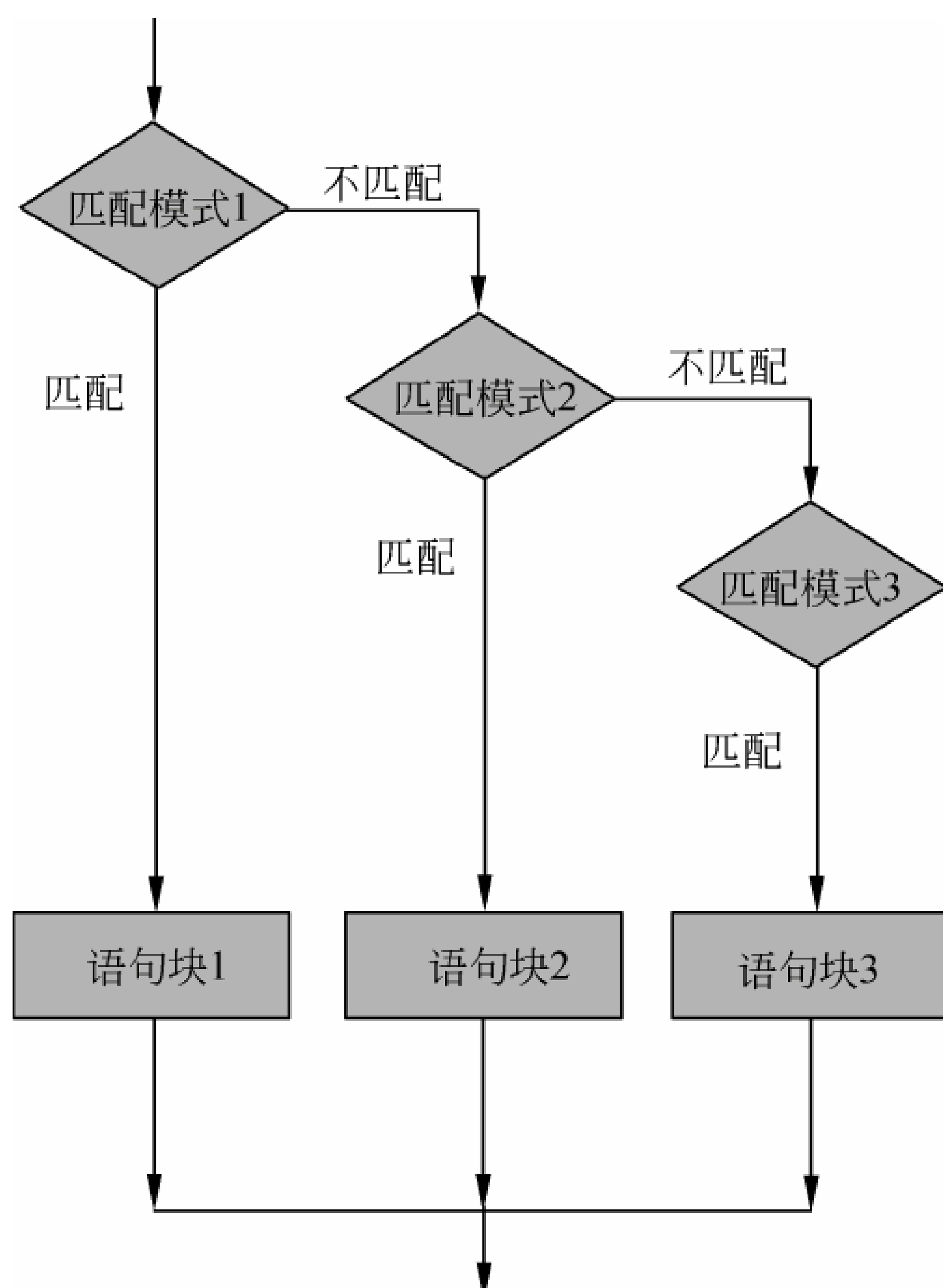


图 18.4　case 语句的执行流程

执行上面的 case 语句时，系统会先判断变量是否与模式 1 匹配，如果匹配，则执行模式 1 后面的语句。如果不匹配，则继续匹配模式 2、模式 3…直到将所有的模式匹配完。格式最后的 esac 是整个 case 语句的结束标记。

在上面的格式中，每个模式后面的两个分号“;;”是模式结束标记。系统执行模式后面的语句时，不会越过此标记。

18.3.2　利用 case 语句处理选项参数

case 语句经常用于检测脚本的选项参数及用户的输入。在本小节中，将使用实例介绍

如何使用 case 语句处理脚本选项参数。

此处引入一个使用 case 处理选项的例子。在下面这个示例脚本中，使用 while 和 case 语句依次处理脚本的所有参数，检测选项并设置变量值。最后根据变量的值启用命令的选项。

查看示例脚本的内容如下：

```
#查看示例脚本的内容
# cat example16.10.sh
#!/bin/bash

#This is a example script
#6/23/11

#定义脚本的选项变量及其值
#选项变量的值为 1，表示选项没有启用
OPTION L=1
OPTION H=1
OPTION A=1
OPTION D=1
FILE NAME="."
#本示例中使用的命令是 ls
CMD="ls"

#如果脚本的参数个数大于 0，就执行循环
while [ $# -gt 0 ]
 do
  #判断位置变量 1 中保存的是哪个参数
  case "$1" in
    #如果位置变量 1 保存的是-l，就将其选项变量设置为 0
    #然后使用 shift 命令移动位置变量
    -l)
      OPTION L=0
      shift
      ;;
   #以下选项 h、a、d 与 l 类似
    -h)
      OPTION H=0
      shift
      ;;
    -a)
      OPTION A=0
      shift
      ;;
    -d)
      OPTION D=0
      shift
      ;;
    #如果位置变量 1 不是上面模式中的任何一个，那么一定是对象参数
    #将对象参数赋值给变量 FILE NAME
     *)
      FILE_NAME=$1
```

```
      shift
      ;;
  esac
done
#上面的语句执行完成后，选项参数对应的变量值都应该为 0
#对象参数则应该保存在变量 FILE_NAME 中

#由于命令 ls 的选项 h 和 l 必须同时使用才生效
#因此如果使用了选项 h，没有使用选项 l，就关闭选项 h
if [ $OPTION H = "0" ]&&[ $OPTION L = "1" ]
  then
    OPTION H=1
fi

#如果同时使用了选项 h 和 l，就设置在 CMD 变量后添加选项 hl
#option h and l
if [ $OPTION H = "0" ]&&[ $OPTION L = "0" ]
  then
    CMD=$CMD" -hl"
fi

#如果只使用了选项 l，并没有使用选项 h，便在 CMD 变量后添加选项 l
#相反的情况已被第 1 个 if 语句排除
#option l
if [ $OPTION L = "0" ]&&[ $OPTION H = "1" ]
  then
    CMD=$CMD" -l"
fi

#如果使用了选项 a，则将其选项添加到变量 CMD 最后
#a option
if [ $OPTION A = "0" ]
  then
    CMD=$CMD" -a"
fi

#如果使用了选项 d，就将选项 d 添加到变量 CMD 最后
#d option
if [ $OPTION D = "0" ]
  then
    CMD=$CMD" -d"
fi

#通过调用 CMD 变量的方式执行变量
$CMD $FILE_NAME
```

上面这个示例脚本使用 ls 命令演示应该如何处理选项参数。在脚本中，先为所有可能的参数都设置一个默认值，然后使用 case 语句处理参数。如果包含参数，就修改变量默认值，最后通过判断变量是否被修改的方式获取用户使用了哪些选项。

执行上述示例脚本：

```
#执行脚本并使用选项 l 和参数/root
# ./example18.10.sh -l /root
total 2199736
drwxr-xr-x 2 root root      4096 May  6 02:16 Desktop
……
#执行脚本并使用选项 l、d 和参数/root
# ./example18.10.sh -l -d /root
drwxr-x--- 22 root root 4096 Sep  2 17:33 /root
```

从上面的示例脚本中可以看出，处理选项参数和对象参数的功能已经实现。

提示： case 语句只能处理较为简单的选项参数，像-ld 这样的参数 case 语句是无法处理的。如果需要处理像-ld、-t 3 这样的参数，可以使用 getopt 或 getopts 命令。由于此命令多用于 C 等编程语言中，并且也不建议在脚本中使用这种较复杂的参数（建议使用参数的方式传递），因此本书并不包括此内容，感兴趣的读者可以阅读相关文档。

18.3.3　利用 case 语句处理用户输入

许多时候需要获得用户的输入，例如执行某个操作需要获得用户的授权时，让用户输入 yes 或 no。像这种需要用户输入固定的值的情况，就可以使用 case 语句来处理。

由于用户输入时，可能会输入相同含义的不同词，例如将 yes 输入成 y、Y、yes、Yes、YES 等，因此可以在模式中使用“|”符号，将多个意义相同的输入组成一个列表，其形式如下：

```
#下面是一个脚本的片断
……
#使用一个无限循环处理用户的输入
while true
  do
    #提示用户输入，并读取用户的输入保存在变量 ANS 中
    echo -n "Please enter yes or no?[yes|no]"
    read ANS
    #使用 case 语句判断用户的输入
    case "$ANS" in
      #如果用户输入的是 y、Y、yes 或 Yes
      y|Y|yes|Yes)
        echo "You enter yes"
        #省略部分是需要执行的功能语句
        ……
        break
        ;;
      #如果用户输入的是 n、N、no 或 No
      n|N|no|No)
        echo "You enter no"
        ……
        break
        ;;
```

```
        #当用户输入不符合要求时，提示用户输入指定的值
        #并使用 continue 语句跳转到 while 语句处重新开始下一个循环
        *)
          echo "Please enter yes or no."
          continue
          ;;
    esac
done
……
```

在上面这个示例中，使用 while 和 case 语句要求用户输入指定的字符串。只有当用户正确输入字符串之后，脚本才进行下一步处理。

这是一个最常见的处理用户输入的例子，用户只能输入指定的值，否则将无法跳出循环。关于其应用的例子将在第 19 章中介绍。

18.4　步进循环语句 for

使用 Shell 编写脚本时，可能会遇到需要处理一个列表中每一个值的情况。例如列表中是服务器需要开放的端口、协议名称等，这时就可以使用步进循环语句 for 来处理。本节将简单介绍 for 循环语句的使用方法。

说明： 在早期的编程语言中，使用 for 语句需要使用 step 语句为其指定步长（步长就是两次取值之间的间隔），因此形象地将其称为步进循环语句。现在大多数编程语言中的 for 语句都不再使用 step 语句，但这个称谓却延续了下来，在本书中也按旧习将其称为步进循环语句。

18.4.1　for 语句的基本格式

使用 for 语句处理列表时，系统会先取得列表中的第 1 个值，然后在循环结构（循环结构是需要重复执行的语句块）中使用第 1 个值。完成后再取得列表中的第 2 个值，第 3 个值……直到将列表中的所有值都取完才停止。

for 循环语句的基本格式如下：

```
for 变量 in 列表
do
   语句块
done
```

上面这个 for 语句的执行流程如图 18.5 所示。

从图 18.5 中可以看出，执行 for 语句时，系统会首先取得列表中第 1 个值，将其赋值给变量，然后再执行语句块（通常会在语句块中使用变量）。执行到 done 语句时，系统会重新跳转到 for 语句开始，并取得列表中的第 2 个值，继续执行 do 语句后面的语句块。系统会重复上面这个过程，直到将列表中的值取完。

18.4.2　利用 for 语句处理数组

数组是一个比较特殊的变量，这是因为在数组中保存的是多个变量组成的列表。本小节将简单介绍如何使用 for 语句处理一个值列表。

（1）为了说明 for 语句的原理，此处引入一个 for 语句的示例脚本 example18.11.sh。在这个脚本中，先定义了一个变量 I，用于计算循环执行的次数。然后使用 for 语句让变量 LOOP 循环地获得列表中的值。每执行一次循环，脚本都会输出循环执行的次数（即变量 I 的值）、变量 LOOP 的值，并将变量 I 的值加 1。

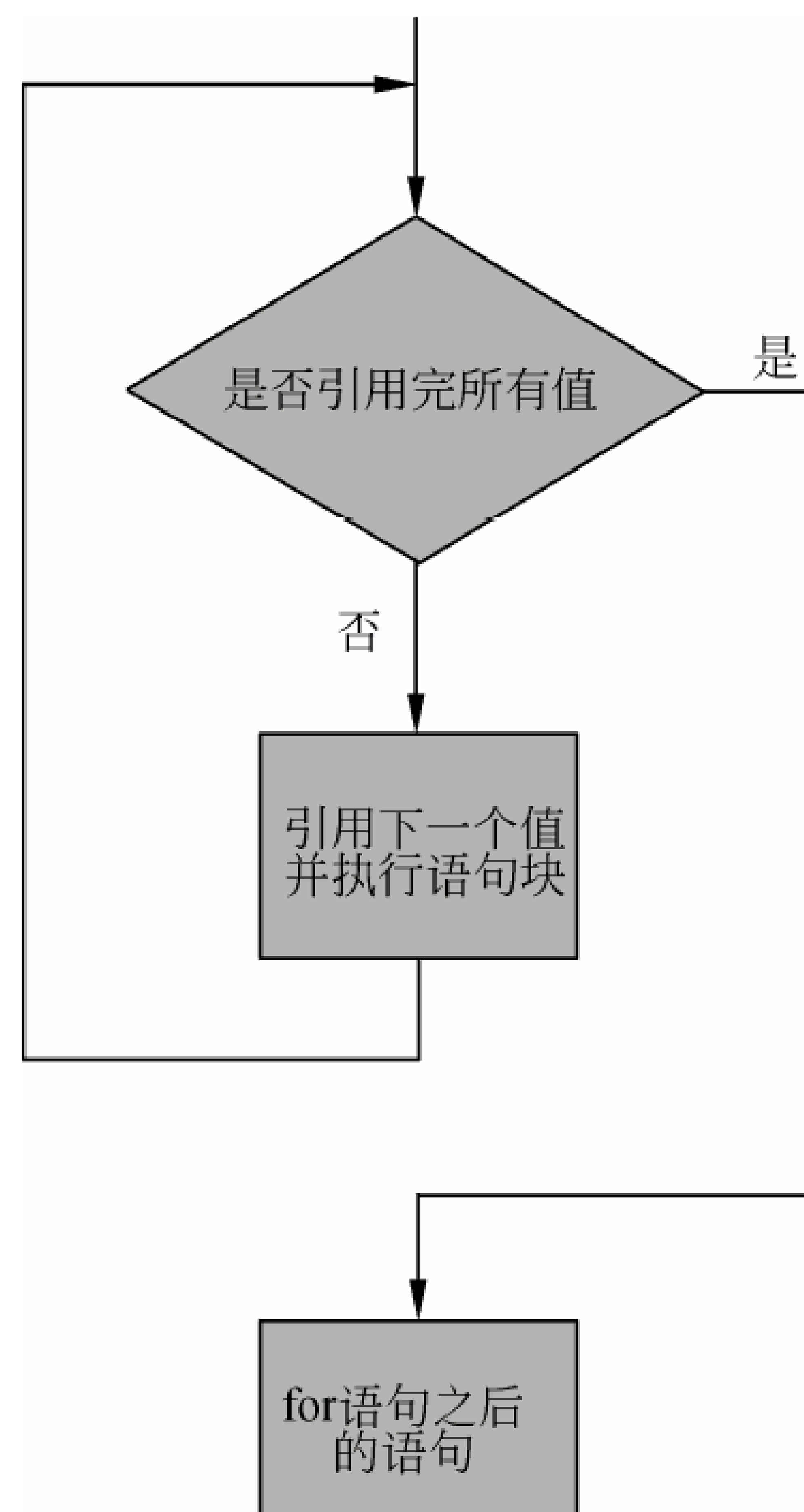


图 18.5　for 语句的执行流程

查看示例脚本的内容：

```
# cat example18.11.sh
#!/bin/bash

#This is a example script
#6/24/11

#定义变量 I，并为其赋值为 1
I=1
#使用 for 语句处理一个值列表
#for 语句会将列表中的每一个数字赋值给变量 LOOP 并执行 do 和 done 之间的语句
for LOOP in 1 2 3 4 5 6
  do
    #显示当前是第几次执行循环
    echo "loop:"$I
    #显示当前循环中变量 LOOP 的值
    echo "LOOP="$LOOP
    #将变量 I 的值加 1
    I='expr $I + 1'
done
exit 0
```

在上面这个脚本中，定义了一个变量 I，用于计算循环的次数，另一个变量 LOOP 则用于循环地获取列表中的值。

执行上面的示例脚本如下：

```
# ./example18.11.sh
loop:1
LOOP=1
loop:2
LOOP=2
loop:3
LOOP=3
loop:4
LOOP=4
```

```
loop:5
LOOP=5
loop:6
LOOP=6
```

从其结果中可以清楚地看出，for 语句先从列表中取第 1 个值并赋值给变量 LOOP，然后执行 do 语句后的语句。当遇到 done 语句时，再取第二个值赋值给变量，依此类推，直到将列表中最后一个值赋值给变量。

（2）当然也可以将上面的脚本改写为数组表示值列表的方式：

```
# cat example18.11.1.sh
#!/bin/bash

#This is a example script
#6/24/11

I=1
#使用数组代替列表
J="1 2 3 4 5 6"
for LOOP in $J
  do
    echo "loop:"$I
    echo "LOOP="$LOOP
    I='expr $I + 1'
done
exit 0
```

由于数组就是一个变量的列表，因此这个脚本与前面的脚本相同。

18.4.3　一个设置防火墙的例子

为了说明 for 语句的应用，此处引入一个使用 for 语句处理防火墙规则的示例脚本 example18.12.sh。在这个示例脚本中，先从用户输入中接收防火墙端口号列表，然后使用 for 语句引用这些端口并使用命令添加规则，最后保存防火墙规则。

查看示例脚本的内容如下：

```
# cat example18.12.sh
#!/bin/bash

#This is a example script.
#6/24/11

#输出提示，要求用户输入允许使用的行程端口号
#接收用户的输入并保存在变量 PORT LIST 中
echo -n "Please enter the port number allowed:"
read PORT LIST

#利用 for 语句处理变量 PORT_LIST 中的端口列表
for PORT in $PORT_LIST
  do
```

```
    #为防火墙添加规则
    iptables -t filter -A OUTPUT -p tcp --dport $PORT -j ACCEPT
    iptables -t filter -A OUTPUT -p udp --dport $PORT -j ACCEPT
    #显示已经添加规则的端口号
    echo "Allowed prot:"$PORT
done

#保存防火墙规则
/etc/init.d/iptables save
exit 0
```

这个脚本的执行结果如下：

```
# ./example18.12.sh
#用户输入要开启的端口号列表
Please enter the port number allowed:80 53 110 25 21
Allowed prot:80
Allowed prot:53
Allowed prot:110
Allowed prot:25
Allowed prot:21
Saving firewall rules to /etc/sysconfig/iptables:        [  OK  ]
```

脚本成功执行，并添加用户输入的端口至防火墙规则。

18.5　循环语句 until

使用 Shell 编写脚本时，许多时候都希望能循环地处理一些信息。例如循环地检测某个文件系统的可用空间，循环地检测与某个重要主机间的通信情况等。因此循环在处理系统监控任务时非常有用。

与其他编程语言一样，在 Shell 脚本中也可以使用循环语句。在 Shell 脚本中常见的循环语句是 until 和 while。本节将简单介绍 until 循环语句。

18.5.1　until 语句的基本格式

until 语句在脚本编程的过程中并不常用，但由于其工作原理特殊，在某些时候使用此语句十分方便。本小节将简单介绍 until 语句的基本格式。

【until 语句的基本格式】

```
until 条件测试
  do
    语句块
done
```

untile 语句的执行流程如图 18.6 所示。

从图 18.6 中可以看出，执行 until 语句时，系统将先执行一次语句 do 和 done 之间的语句块，然后再进行条件测试。如果条件测试为假，则继续执行 do 和 done 之间的语句块，

直到条件测试为真时，until 语句执行结束。

【until 语句的工作原理】

为了讲解 until 的工作原理，此处引入一个示例脚本 example18.13.sh。在这个示例脚本中，until 语句会先判断变量 I 的值是否大于等于 5。如果不满足，就将 I 的值加 1，然后输出 I 的值。最后再判断变量 I 的值是否大于等于 5，直到满足为止。

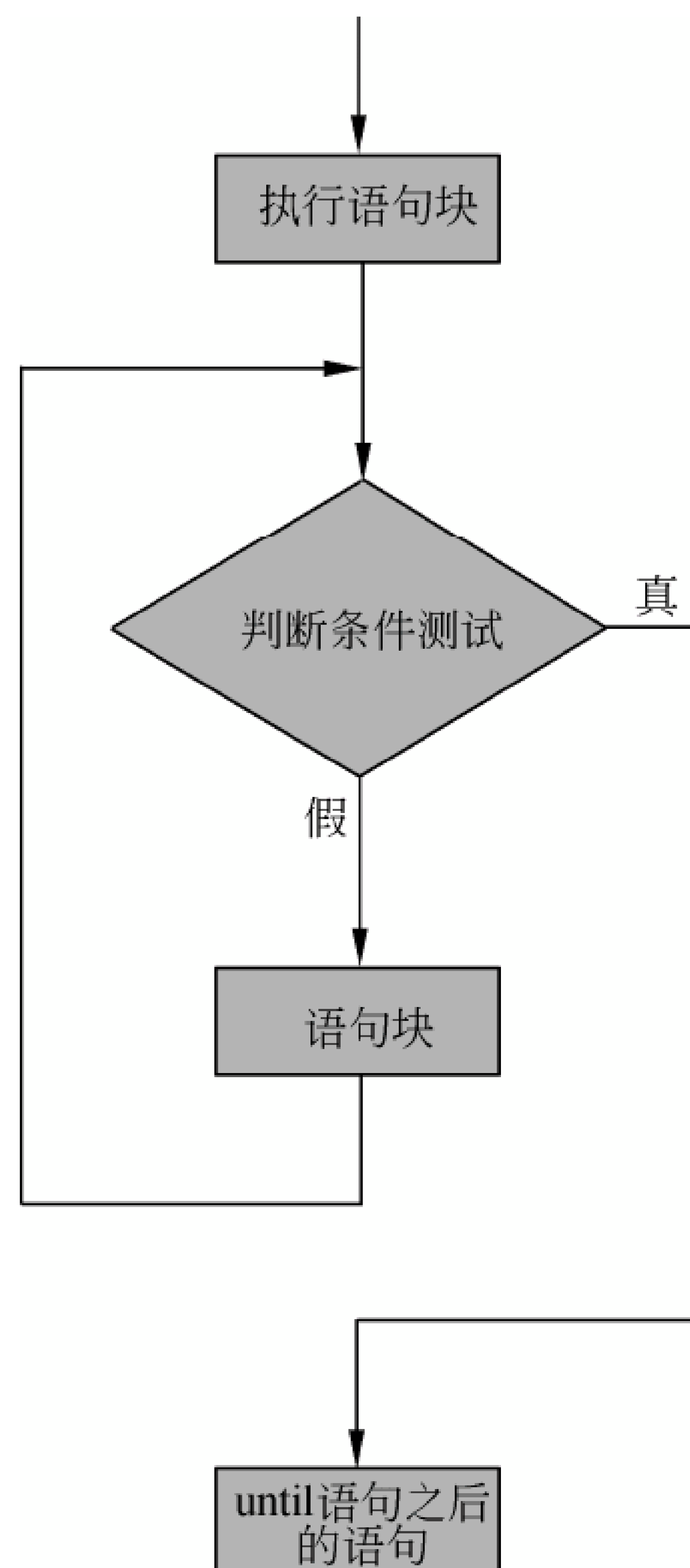


图 18.6　until 语句的执行流程

查看示例脚本的内容如下：

```
# cat example18.13.sh
#!/bin/bash

#This is a example script.
#6/24/11

#定义变量 I，并为其赋值为 0
I=0

#使用 until 循环，循环的出口为变量 I 的值大于等于 5
until [ $I -gt 5 ]
  do
    #将变量 I 的值加 1
    I='expr $I + 1'
    #输出变量 I 的值
    echo "I="$I
done
```

运行这个示例脚本内容如下：

```
# ./example18.13.sh
I=1
I=2
……
I=5
I=6
```

从上面这个脚本的执行结果可以看出，如果 until 语句中的条件测试失败，则执行 do 和 done 语句之间的语句。如果条件测试为真，就跳出循环，执行 done 之后的语句。

18.5.2　利用 until 语句监控文件系统状态

编写脚本 Shell 时，经常需要使用一种无限循环式的结构，例如利用无限循环一直不停地检测系统网络连接、监控系统磁盘使用率等。本小节将介绍利用 until 语句的循环结构，编写一个脚本实现监控文件系统状态的功能。

使用 until 语句条件测试为真时退出的特点，编写监控系统状态的脚本是一个好办法。此处引入一个使用 until 语句编写的监控系统文件系统的示例脚本 example18.14.sh。在这个示例脚本中，先使用 df、grep、awk 和 sed，这 4 个命令获取文件系统/dev/md0 的使用率，然后使用 until 循环语句判断使用率是否大于等于 90%。如果文件系统/dev/md0 的使用率大

于等于 90%，则发送邮件给 root。当文件系统的使用率大于等于 95%时，由于 until 的条件测试为真，因此会自动退出循环。

查看示例脚本的内容：

```
# cat example18.14.sh
#!/bin/bash

#This is a example script
#6/24/11

#使用 df、grep、awk、sed 命令获取文件系统 md0 的使用率，并保存在变量 USED 中
USED='df -k | grep "/dev/md0" | awk '{print $5}' | sed 's/%//''

#循环的跳出条件为文件系统的使用率大于等于 95%
until [ "$USED" -gt "95" ]
  do
   #获取文件系统 md0 的使用率，并保存在变量 USED 中
   USED='df -k | grep "/dev/md0" | awk '{print $5}' | sed 's/%//''
   if [ $USED -gt "90" ]
     then
       #当使用率大于等于 90%时，将使用情况保存在临时文件中
       echo "disk:/dev/md0" >/tmp/disk.tmp
       echo "Used:"$USED >>/tmp/disk.tmp
       #将临时文件中的内容作为邮件内容发送给 root 用户，并删除临时文件
       mail -s "disk warning!" root </tmp/disk.tmp
       rm -rf /tmp/disk.tmp
   fi
   #暂停 10 分钟
   sleep 600
done
```

需要注意的是在这个脚本中，使用了一个 sleep 命令，每次执行完成后，进程将处于睡眠状态 10 分钟，以免循环消耗掉过多的系统资源。

注意：任何循环语句都应该小心控制（特别是在无限循环中），避免循环消耗掉过多的系统资源。

18.6　while 循环语句

在 Shell 脚本编程中，如果要使用循环，大多数情况下都不会使用上一节中介绍到的 until 语句，多数情况下都会使用 while 语句。while 循环语句是一个非常常用的循环语句，其原理与 until 正好相反。本节将简单介绍 while 语句及使用方法。

18.6.1　while 语句的基本格式

while 语句是一个与 until 原理相反的循环语句，其基本格式如下：

```
while 条件测试
  do
     语句块
done
```

while 语句的基本格式与 until 语句类似，但原理却完全相反。while 语句的执行流程如图 18.7 所示。

从图 18.7 中可以看出，执行 while 语句时，系统会首先判断条件测试是否为真，如果条件测试的结果为真，则执行 do 和 done 语句之间的语句块。执行完成后继续判断条件测试是否为真，直到条件测试为假时退出循环，并执行 done 之后的语句。

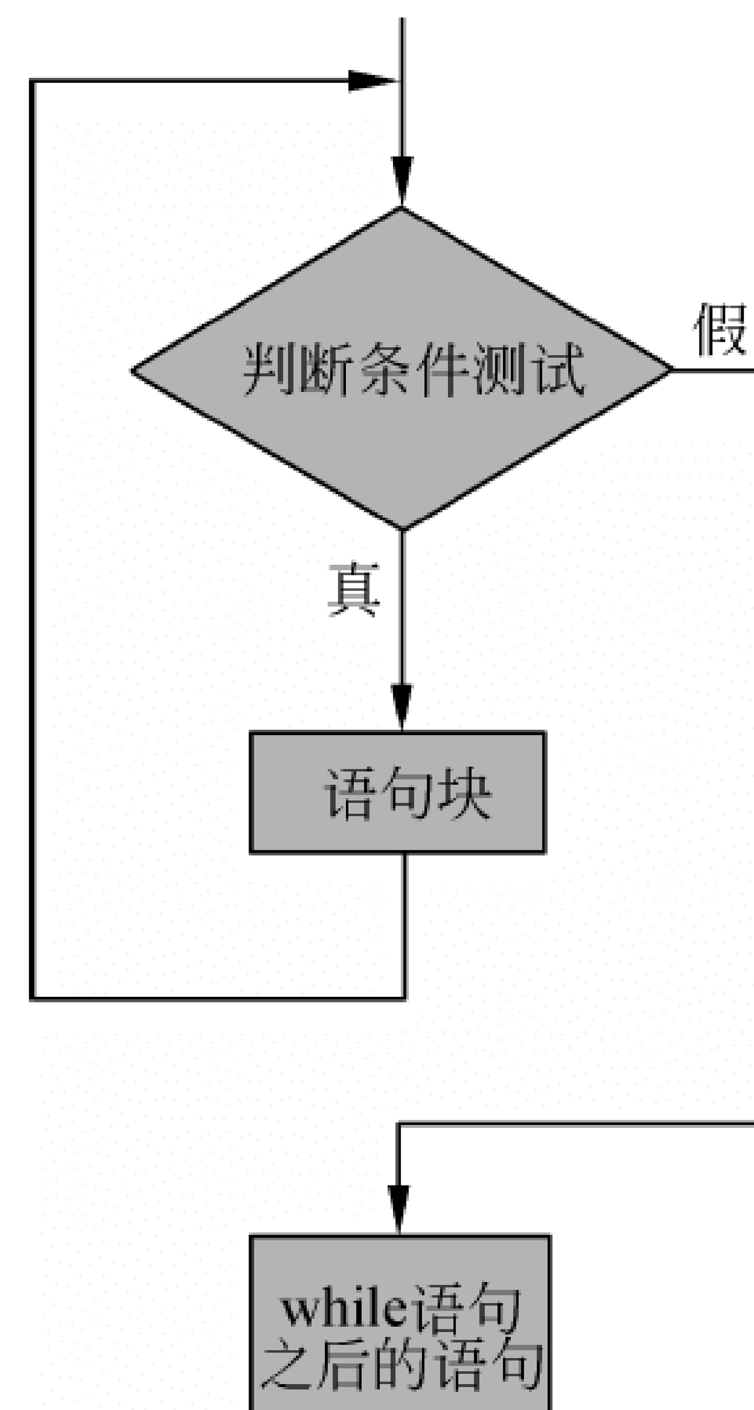


图 18.7　while 语句的执行流程

18.6.2　while 语句与 until 语句的区别

为了说明 while 语句与 until 语句间的区别，此处引入一个示例脚本 example18.15.sh。这个示例脚本与前面的示例 example18.13.sh 相似，不同的是，此处使用的条件是小于等于。

查看示例脚本的内容：

```
# cat example18.15.sh
#!/bin/bash

#This is a example script.
#6/24/11

#定义变量 I 并赋值为 1
I=1
#使用 while 语句，并设置循环出口为变量 I 的值小于等于 6
while [ $I -lt 6 ]
  do
   #显示循环执行的次数
   echo "loop:"$I
   #显示变量 I 的值
   echo "I="$I
   #将变量 I 的值加 1
   I='expr $I + 1'
done
```

运行这个脚本查看其运行结果如下：

```
# ./example18.15.sh
loop:1
I=1
loop:2
I=2
loop:3
```

```
I=3
loop:4
I=4
loop:5
I=5
```

这个脚本的运行结果充分地说明了 while 语句与 until 语句的区别。

- while 语句先判断条件测试是否为真，如果为真，再执行循环结构中的语句；until 语句则先执行一次循环结构，再判断条件测试是否为真。
- 在 while 语句中，如果条件测试为假，系统就会跳出 while 语句；在 until 语句中，只有条件测试为真时才跳出循环。

在脚本中使用 while 语句和 until 语句时，应该特别注意以上区别，以免写错条件测试，出现无限循环的情况。

18.6.3　利用 while 语句监控系统网络状态

除了上一节中介绍的 until 语句以外，while 语句也可以用于持续对系统状态进行监控。下面引入一个示例脚本 example18.16.sh 作说明。这是一个使用 ping 命令判断与目标主机连接状态的监控脚本。其中 while 语句的条件测试被设置为 true，这表示 while 语句将会一直运行，直到用户对其使用 kill 命令为止。

在这个脚本中，先使用 ping 命令测试与目标主机的连接状态。如果测试失败，就一一测试与网关的连通性、数据包经过路由器的连通性等，并将结果写入日志。

查看示例脚本的内容如下：

```
# cat example18.16.sh
#!/bin/bash

#This is a example script.
#6/24/11

#设置用于保存网络状态和路由状态的日志文件变量
NET STATUS LOG="/root/netstatus.log"
NET TRACEROUTE LOG="/root/net traceroute.log"

#使用 true 启用一个无限循环
while true
  do
    #将当前的日期、时间保存在变量 DATE 中
    DATE='date +"%Y-%m-%d %H:%M:%S"'
    #使用 ping 命令发送 3 个数据包，测试与 192.168.118.91 的连通性
    ping 192.168.118.91 -c 3 &>/dev/null
    #如果测试成功，就将当前日期、时间和网络状态写入网络状态日志文件中
    if [ $? = 0 ]
      then
        echo "$DATE     YES" >>$NET STATUS LOG
    else
        #如果测试失败，就写入日志，并测试与网关之间的连通性
```

```
        echo "$DATE    NO" >>$NET STATUS LOG
        ping 192.168.115.1 -c 3 &>/dev/null
        #使用 if 语句判断与网关之间的连通性
        if [ $? = 0 ]
          then
            #如果网关测试通过，就将日志和 traceroute 命令的输出保存在日志文件中
            #以便管理员排除错误
            echo "################$DATE Begin################" >>$NET
            TRACEROUTE LOG
            echo "traceroute:" >>$NET TRACEROUTE LOG
            traceroute 192.168.118.91  >> $NET TRACEROUTE LOG
            echo "################$DATE End################" >>$NET
            TRACEROUTE LOG
        else
            #如果网关测试失败，就写入日志中
            echo "Disconnected with the default gateway." >>$NET STATUS LOG
        fi
    fi
    #设置测试间隔为 60 秒
    sleep 60
done
```

当管理员不能实时监控网络时，这个脚本就可以代替管理员监控网络，并将可用于排错的信息写入日志中。

读者可以通过完善这个脚本，学习如何使用 Shell 编程管理系统。例如将日志内容通过邮件发送给管理员，添加更多测试内容等。

18.7 利用 break 和 continue 控制循环

在 Shell 脚本中，循环能带给编程者巨大的便捷，然而有时问题也随之出现。例如不能随意跳过循环中的特殊步骤，循环过程中无法跳出等。这时就可以使用两个比较特殊的语句控制循环：break 和 continue 语句。本节将简单介绍如何使用 break 和 continue 语句控制循环。

18.7.1 使用 break 语句控制循环

break 的功能是跳出循环语句。当循环不需要继续执行时，可以使用 break 语句立即跳出循环。

（1）为了说明 break 语句的使用，此处引入一个示例脚本 example18.17.sh。在这个示例脚本中，使用了一个无限循环的 while 语句，然后配合使用 if 和 break 语句重新为这个循环语句 while 制作了一个出口。当变量 I 的值大于 6 时，break 语句将会跳出循环语句 while。

查看示例脚本的内容如下：

```
# cat example18.17.sh
```

```
#!/bin/bash

#This ia a example script
#6/24/11

#定义变量 I 并赋值为 1
I=1

#定义一个无限循环
while true
  do
   if [ $I -gt 6 ]
     then
       #当变量 I 的值大于 6 时，使用 break 跳出循环
       break
   fi
   #显示变量 I 的值，并将变量 I 的值加 1
   echo "I="$I
   I='expr $I + 1'
done
```

运行上面的示例脚本如下：

```
# ./example18.17.sh
I=1
……
I=6
```

从脚本的输出中可以看到，使用 if 和 break 语句重新制作的出口生效了。

（2）如果使用了多个循环语句嵌套执行，还可以指定要 break 跳出的循环数。为此修改上面的示例脚本，使用 break 2 来跳出两个循环。

查看修改后的示例脚本内容如下：

```
# cat example18.17.1.sh
#!/bin/bash

#This ia a example script
#6/24/11

#定义变量 I 和 J，并分别赋值为 1
I=1
J=1
#定义两个无限循环
while true
  do
   while true
     do
       if [ $J -gt 3 ]
         then
           #使用 if 语句判断，当变量 J 的值大于 3 时，使用 break 语句跳出两个无限循环
           break 2
       fi
```

```
        #显示变量 I 和 J 的值
        echo "I="$I
        echo "J="$J
        #变量 J 的值加 1
        J='expr $J + 1'
    done
    #变量 I 的值加 1
    I='expr $I + 1'
done
```

运行上面的脚本如下：

```
# ./example18.17.1.sh
I=1
J=1
……
I=1
J=3
```

从上面的结果可以看出，使用了参数 2 的 break 一次跳出了两个嵌套的循环语句。

18.7.2　使用 continue 语句控制循环

与前面的 break 语句不同的是，continue 语句并不会退出循环，而是跳过当前循环，开始下一次循环过程。可以理解为当遇到 continue 语句时，系统将会重新执行条件测试并开始下一次循环。

为了说明 continue 语句的使用方法，此处引入一个示例脚本 example18.18.sh。在这个示例脚本中，使用 continue 语句来跳过循环次数为 3（判断变量 J 的值是否等于 3）的循环过程。

查看示例脚本的内容如下：

```
# cat example18.18.sh
#!/bin/bash

#This ia a example script
#6/24/11

#定义变量 I 和 J，并赋值为 1
I=1
J=1
#定义两个无限循环
while true
  do
   while true
     do
       if [ $J = 3 ]
         then
           #当变量 J 的值等于 3 时，使用 continue 跳过当前循环，执行下一次循环
           echo "use continue"
           #当变量 J 大于 3 时其值加 1
```

```
            J='expr $J + 1'
            continue
        fi
        if [ $J -gt 4 ]
          then
            #当变量 J 的值大于 4 时，使用 break 跳出两个循环
            break 2
        fi
        #显示两个变量的值，并将变量 J 的值加 1
        echo "I="$I
        echo "J="$J
        J='expr $J + 1'
    done
    #变量 I 的值加 1
    I='expr $I + 1'
done
```

运行这个脚本如下：

```
# ./example16.18.sh
I=1
……
J=2
use continue
I=1
J=4
```

从上面的输出结果可以看出，执行到 continue 语句时，系统将会跳过没有执行完的语句，并开始测试条件重新执行下一个循环过程。

提示： 在 Shell 编程中，break 和 continue 语句与 until 一样，使用非常少但在有时却非常有用，因此建议初学者了解其使用方法即可。

18.8　小　　结

- ❑ 18.1 节主要介绍了 Shell 脚本中的条件测试和信号捕获。相对于其他语言而言，这种测试方法比条件表达式更简单、易懂。条件测试在编程中使用十分普遍，因此大多数初学者应该掌握这些内容。
- ❑ 18.2 节主要介绍了 if 语句。if 语句是一个十分常用的流程控制语句，其形式多样（可以使用 if、if else、if elif 等）且使用十分灵活，在实际编程中经常用到。建议初学者掌握这个语句的用法。
- ❑ 18.3 节介绍了脚本编程中的 case 语句。这个语句通常用来处理用户的输入，因此初学者应该了解这个语句的用法。
- ❑ 18.4 节讲解了步进循环语句 for。for 语句是一个不太常用的语句，读者理解这个语句即可。

- 18.5 节介绍了 until 循环语句的使用方法。until 在 Shell 中是一个不太常用的循环语句，读者了解其使用方法即可。
- 18.6 节简单介绍了 Shell 中最常用、最容易理解的 while 语句。它在 Shell 脚本编程中应用十分广泛，经常用来处理系统监控、循环调用等问题，初学者应该掌握这个语句的用法。同时还需要注意 while 语句与 until 语句的区别。
- 18.7 节介绍了 Shell 中用于控制循环的 break 和 continue 语句。这两个语句是对流程控制的重要补充，但实际中较少使用。对于初学者而言，理解其用法即可。

流程控制是对 Shell 脚本编程的重要补充，它让 Shell 脚本“拥有”了复杂的判断的功能。本章介绍的流程控制语句和示例脚本中，初学者应该掌握利用脚本监控系统的方法。

第 19 章　Shell 编程技巧和应用实例

通过前面几章的学习，读者可能会发现编写 Shell 脚本其实就是将常用的命令通过各种语句有机地组合起来，以实现更复杂的功能。为了使读者能更好地将各种语句有机地组合起来，在本章中将介绍一些常见的脚本编程技巧、调试方法，使用实例介绍 Linux 系统中管理脚本的编写方法和思路等。

本章涉及的主要内容如下。

- 介绍运行脚本的相关知识，如何编写一个运行脚本及向系统添加服务。
- 介绍如何编写好 Shell 脚本，在脚本编写过程中应该注意的事项、编程习惯，以及 Shell 脚本的安全性等。
- 介绍脚本调试的方法及排除脚本中错误的方法。
- 介绍使用脚本管理系统的实例及如何修改、管理脚本。
- 介绍系统管理脚本的编写思路、方法等。

19.1　运行级别脚本

在前面的章节中，介绍了 Linux 系统中的运行级别及系统服务相关概念。其中介绍了关于运行级别脚本的概念，运行级别脚本是开启、关闭系统服务的脚本。在本节中将介绍运行级别脚本的结构，以及如何编写运行级别脚本。

19.1.1　运行级别脚本的结构

在编写运行级别脚本之前，需要了解运行级别脚本的结构。为此可以查看系统中已经存在的系统脚本，本例中将使用 NTP 服务的运行级别脚本/etc/init.d/ntpd 作为示例：

```
# cat /etc/init.d/ntpd
#!/bin/bash
#
# ntpd          This shell script takes care of starting and stopping
#               ntpd (NTPv4 daemon).
#
#下面这句中的-表示默认不启动
#58 和 74 分别表示系统启动和退出时启动、关闭服务的优先级
# chkconfig: - 58 74
#以下为运行级别脚本的描述信息
# description: ntpd is the NTPv4 daemon. \
……
```

```
#省略部分为读取配置文件相关内容
#定义启动函数 start
start() {
        #检查当前系统网络是否开启
        #如果未开启，则退出脚本
        # Check that networking is up.
        [ "$NETWORKING" = "no" ] && exit 1

        #调用 readconf 函数读取配置文件
        readconf;

        #同步系统时钟
        if [ -n "$dostep" ]; then
            echo -n $"$prog: Synchronizing with time server: "
            /usr/sbin/ntpdate $dropstr -s -b $NTPDATE OPTIONS $tickers
            &>/dev/null
            ……
        fi
        #启动 NTP 服务的守护进程
        # Start daemons.
        echo -n $"Starting $prog: "
        daemon ntpd $OPTIONS
        RETVAL=$?
        echo
        #如果启动 NTP 服务成功，就创建服务启动标志文件
        [ $RETVAL -eq 0 ] && touch /var/lock/subsys/ntpd
        return $RETVAL
}

#定义停止服务函数 stop
stop() {
        #关闭服务
        echo -n $"Shutting down $prog: "
        killproc ntpd
        RETVAL=$?
        echo
        #如果关闭服务并删除标志文件
        [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/ntpd
        return $RETVAL
}

#下面是判断参数 1 的结构
# See how we were called.
case "$1" in
  #如果参数 1 是 start，就调用启动函数 start
  start)
        start
        ;;
  #如果参数 1 是 stop，就调用停止服务函数 stop
  stop)
        stop
        ;;
```

```
  #如果参数 1 是 status，就使用 status 判断 ntpd 的状态
  status)
        status ntpd
        RETVAL=$?
        ;;
  #如果参数 1 是 restart 或 reload，就调用停止、启动函数重新启动服务
  restart|reload)
        stop
        start
        RETVAL=$?
        ;;
  #如果参数 1 是 condrestart，就判断文件标志并重新启动服务
  condrestart)
        if [ -f /var/lock/subsys/ntpd ]; then
            stop
            start
            RETVAL=$?
        fi
        ;;
  #如果参数 1 不是以上参数，则输出提示信息
  *)
        echo $"Usage: $0 {start|stop|restart|condrestart|status}"
        RETVAL=3
esac

#设置退出状态并退出
exit $RETVAL
```

这是一个非常长的脚本，上面仅截取了该脚本的一部分，此部分正好说明了运行级别脚本的编写方式。

- 首先需要以注释的方式声明服务默认启动的运行级别列表，系统启动、关闭时服务启动、关闭的优先级。
- 通常将服务的启动、关闭操作都写在函数中，然后以函数的方式调用。
- 使用 case 语句处理 start、stop 等参数。
- 在 case 语句最后处理不合适的参数。
- 如果需要检测服务是否处于运行状态，最好创建运行标记文件。

当然也可以不按照上面的方法编写，例如当启动、关闭操作的语句都较少时，也可以不写入函数等。了解了以上内容之后，就可以自己动手编写运行级别脚本。

注意：设置运行级别脚本启动、关闭的优先级时，一定要注意其依赖的其他服务的优先级，以免启动服务失败。

19.1.2　编写运行级别脚本

编写运行级别脚本的目的可能是为自己编写的监控脚本添加新的启动方式，也可能是为某个产品编写启动服务等。此处引入一个示例脚本：

```
# cat test
#!/bin/bash

#定义服务启动的运行级别为 345
#启动和关闭的优先级为 80、10
# chkconfig:345 80 10
#以下这句为服务的描述信息
# description:This is a test service

#This is a test script
#6/25/11

#定义参数错误的提示消息函数 usage
function usage()
{
   echo "Usage:$0{start|stop|restart|reload}"
   return 0
}

#定义启动服务的函数 start
function start()
{
   echo "Starting $0:"
   return 0
}

#定义关闭服务的函数 stop
function stop()
{
   echo "Shutting down $0:"
   return 0
}

#脚本主体部分
#使用 case 语句判断参数的值
case $1 in
  #如果参数 1 为 start，就调用 start 函数
  start)
     start
     ;;
  #如果参数 1 为 stop，就调用 stop 函数
  stop)
     stop
     ;;
  #如果参数 1 为 restart 或 reload，就调用 stop 和 start 函数重新启动服务
  restart|reload)
     stop
     start
     ;;
  #如果参数 1 不是以上值，就调用 usage 函数，并设置退出状态退出脚本
  *)
     usage
```

```
    exit 1
esac
exit 0
```

读者需要注意“# chkconfig”和“# description”开头的这两行，这是运行级别脚本中须定义的内容，主要用于添加服务时初始化配置和描述信息。

上面这个示例运行级别脚本中，并没有包括任何实质性的内容。在实际编写运行级别脚本时，需要在函数中添加实际的启动、关闭服务的语句。在目录/etc/init.d 中还有许多运行级别脚本，读者可以通过研究这些脚本，进一步学习如何编写运行级别脚本。

注意：读者也可以从互联网上下载有用的软件，例如 Apache（提供 Web 服务的服务器软件）。编译安装后，手动添加这些软件的运行级别脚本。

19.1.3　添加和管理运行级别脚本

编写运行级别服务脚本的目的是能够将脚本作为服务添加到系统中。本小节将简单介绍如何将运行级别脚本添加为系统服务并管理系统服务。

1．添加运行级别脚本为系统服务

将运行级别脚本添加为系统服务之前，应该将编写完成的运行级别脚本复制到目录/etc/init.d 中，并添加相应的执行权限。完成之后，就可以使用 chkconfig 命令将运行级别脚本添加为系统服务了。

将示例脚本 test 添加为系统服务使用如下命令：

```
#使用 chkconfig 命令的 add 选项添加系统服务
# chkconfig --add test
```

添加成功后，命令不会有任何返回信息。可以使用以下命令查看添加的服务：

```
#使用 chkconfig 命令查看添加的系统服务
# chkconfig --list test
test           0:off    1:off    2:off    3:on     4:on     5:on     6:off
#使用 chkconfig 和 grep 命令查看添加的系统服务
# chkconfig --list | grep test
test           0:off    1:off    2:off    3:on     4:on     5:on     6:off
```

2．管理使用运行级别脚本添加的服务

使用 chkconfig 命令添加的服务之后，就可以使用命令 service 启动服务：

```
#启动新添加的服务
# service test start
Starting /etc/init.d/test:
```

当然也可以随时删除：

```
#使用 chkconfig 命令删除服务 test
# chkconfig --del test
```

这样添加的服务 test 就已经被删除掉了。

19.2　怎样写好 Shell 脚本

对于许多初学者而言，编写一个可以运行的脚本可能需要花费许多时间。这些时间可能大多用于构思脚本的主体结构、排除脚本中的错误等。本节将简单介绍编写 Shell 脚本的基本步骤、基本原则，以及脚本的安全性等内容。

19.2.1　一般性原则

为了帮助初学者快速熟悉 Shell 脚本的编写方法，本小节将简单介绍编写脚本的基本原则和特殊技巧。

1．脚本编写步骤

许多时候编写的脚本功能和结构都非常复杂，脚本可能会使用许多复杂的计算过程、判断语句、循环语句等。在编写这类功能较为复杂的脚本时，应该从最基本的函数开始。先编写要使用的函数，每完成一个函数，就对函数进行详细测试，以确认函数能够满足实际需要。当然测试函数的同时，就可以写出部分脚本主体内容。待所有函数都编写完成，并确认没有问题之后，再逐渐添加脚本的主体内容，最终编写脚本。

在这个过程中，每编写一个小的功能，都需要对新编写的功能进行测试。完善其功能之后，再编写下一个功能，直到脚本完成。

2．模块化脚本

Linux 系统中的许多应用程序都被模块化，这些软件被划分为一个个很小的功能模块。这是一个非常好的主意，其最大的好处是对软件进行维护时，不必大动干戈地修改整个应用软件，只需要修改个别的功能模块即可。

编写脚本时，特别是编写一些功能较复杂的脚本时，也应该采用这种思想，将复杂的脚本功能模块化。脚本模块化的通常做法是将复杂的脚本功能分解为多个函数，然后在脚本的主体部分中调用函数实现功能。

如果需要在一台主机上运行多个功能复杂的脚本，可以将这些脚本需要使用的函数放在函数文件中。然后在需要使用函数的脚本中，引用函数文件并调用函数即可。

在实际编写脚本过程中，还有许多需要注意的事项。这在本书的前面几个章节中已经介绍过，此处不再一一赘述。

19.2.2　良好的编程习惯

一个良好的编程习惯不仅在调试脚本时，能更快地排除错误，也可以为后期脚本维护带来便利。因此有必要在学习之初，就养成良好的习惯。本小节将简单介绍编程中的良好

习惯。

1．清晰的结构

为了方便阅读和后期维护，编写的脚本一定要有非常清晰的结构。与其他编程语言不同，在 Shell 脚本中，变量不必定义就能直接使用。因此如果在脚本中，随处都是定义变量、函数的语句，脚本的结构将会变得非常混乱，这非常不利于阅读和维护。

为了解决这个问题，通常建议在脚本开始处集中定义变量和函数。集中定义时，推荐的顺序是：捕获系统信号，定义变量，定义函数。集中定义后面是脚本的主体部分，最后是清除变量、函数的语句。

提示：不同的函数之间，应该使用空行隔开，以表明这些语句分属于不同的功能模块。

2．详细的注释

在脚本中加上详细的注释，可以大大提高脚本的可读性，也可以为后续的脚本维护工作省去许多麻烦。通常建议添加注释的地方如下。

- 在脚本的开始处添加脚本的用途、创建的时间等注释信息。
- 如果有必要，还需要在脚本开始处添加脚本作者、维护信息，以及版权信息等注释内容。
- 为函数、特殊的变量及复杂的语句添加详细的注释。

注释在脚本中十分重要，通常建议在脚本中添加更多的注释，以便阅读和维护脚本时使用。

19.2.3　Shell 脚本的安全性

Shell 脚本给管理和使用系统都带来了巨大的便利，然而在多用户系统中，也带来了许多安全性的问题。在脚本中可能存在的安全性问题如下。

- 在脚本中使用 cd 命令，切换到不安全的目录执行脚本。
- 在脚本中修改了环境变量的值，从而导致系统产生了变化。
- 在某个目录中创建了非法的文件。例如使用重定向在目录/etc 中创建了文件 nologin，这将导致其他用户无法登录。

这些安全性问题产生的后果可能会非常严重，因此有必要防范这些不安全的因素，以免为系统留下安全隐患。

在 Shell 脚本中，提供了一种受限制模式。脚本在受限模式中运行时，可以极大地保护系统的安全性。当脚本中出现 cd 命令、重定向、修改环境变量等不安全的行为时，Bash 将会拒绝执行。

1．调用Shell时启动受限模式

要启动 Bash 的受限制模式，可以在调用 Shell 语句后加上选项 r，这时如果脚本中出现不安全的命令，会被系统拒绝。为了说明受限模式的使用，此处引入一个示例脚本 example19.1.sh。在这个示例脚本中，将验证在受限模式中的切换目录、重定向、修改环境

变量等功能是否被禁用。

查看示例脚本的内容如下：

```
# cat example19.1.sh
#!/bin/bash -r
#上面的选项 r 表示开启受限模式

#this is a example script
#6/25/11

#使用 cd 命令将目录切换到根目录
cd /
#然后使用 pwd 命令验证结果
echo "'pwd'"

#使用 echo 命令和重定向写入文件
echo "test" >~/test.tmp
cat ~/test.tmp

#修改环境变量 SHELL 的值，然后验证
SHELL=/bin/ksh
echo "SHELL="$SHELL
```

执行上面的脚本如下：

```
# ./example17.1.sh
#下面是切换工作目录命令 cd 执行失败的消息
./example17.1.sh: line 6: cd: restricted
#从 pwd 命令的结果可以看出工作目录未改变
/root/example
#下面是重定向失败的消息
./example17.1.sh: line 9: /root/test.tmp: restricted: cannot redirect output
#cat 命令执行失败，其原因是文件不存在
cat: /root/test.tmp: No such file or directory
#下面是修改环境变量命令失败的消息
./example17.1.sh: line 12: SHELL: readonly variable
SHELL=/bin/bash
```

从上面的执行结果可以看出，在 Shell 的受限制模式下切换工作目录、重定向及修改环境变量等操作都被拒绝。

2. 使用set命令启动受限模式

除了调用 Shell 时启动受限模式之外，还可以使用 set 命令的选项 r 启动 Bash 的受限制模式。修改上面的示例脚本，使用 set 命令启动受限模式：

```
# cat example19.1.1.sh
#!/bin/bash

#this is a example script
#6/25/11
```

```
#使用 set 命令的选项 r 启动受限模式
set -r

#使用 cd 命令切换工作目录至根目录
cd /
……
```

执行脚本其结果如下：

```
# ./example19.1.1.sh
./example17.1.sh: line 8: cd: restricted
/root/example
./example17.1.sh: line 11: /root/test.tmp: restricted: cannot redirect
output
……
```

从上面的结果可以看出，使用 set 命令的选项 r 也可以启动受限模式。

3．临时文件的安全性

除了受限制模式之外，如果在脚本中使用了临时文件，这些临时文件也可能会造成安全性问题。为此建议不要将临时文件放入系统临时目录/tmp 中，因为任何登录系统的用户都可以看见系统临时目录中的文件。除此之外，脚本运行完成或由系统产生的中断退出时，建议删除脚本使用的临时文件。

注意：本小节仅简单讨论了 Shell 脚本及使用过程中可能产生的安全性问题，感兴趣的读者可以查阅相关文档，以获取更多信息。

19.3　调 试 脚 本

对于许多初学者而言，调试脚本可能是一件非常恼火的事情，花费了许多时间仍然无法排除脚本中的错误，甚至花费了许多时间连一点头绪都没有。这些错误可能由以下原因引起。

- 一不小心，漏写重要的符号或引用符，导致逻辑错误。
- 将某个命令的用法或将命令本身写错，从而导致脚本的中间过程发行错误。
- 脚本运行时没有提示任何错误，但却没有得到预期的结果。这种情况一般是由逻辑错误引起的。

在学习初期，这些问题会困扰初学者很长时间，因此有必要讲解如何从脚本中排除错误、调试脚本等。在本节中将结合初学者在编写脚本过程中出现的问题，简单介绍如何排除错误并调试脚本。

19.3.1　排除错误

人总是会出错的，在编写脚本的过程中，一个小小的错误都有可能导致脚本无法运行，

通常这都是人为的输入错误。也可能是设计某个流程时，一时大意，将其中某几个步骤弄乱，造成不可理解的错误等。

与其他编译器的错误提示相比，Shell 的错误提示信息经常会让人感到莫名其妙。这是因为 Shell 总是告诉编写者，在某一行或某个符号附近发生了错误，而不是具体的语句。为了帮助初学者快速掌握 Shell 编程的方法，本小节列举并简单介绍编写脚本的过程中经常出现错误的例子。

1. 输入错误

在 Shell 编程中，常见的输入错误如下。

- 输入时不慎将大小写输入错误，这可能会导致一些错误。这是因为 Shell 脚本中的变量、函数、语句等都区分大小写。为了避免大小写错误，建议脚本中的变量都使用大写字母，函数都使用小写字母和下划线表示。
- 由于变量可以不定义就使用，因此当用户输入的变量错误时，可能会得不到任何提示，需要特别注意。

除此之外，还有其他一些内容也容易输入错误，例如命令、命令的选项等。但这都是比较容易排除的错误，此处不仔细介绍。

2. 引用符号

引用符号在脚本中经常使用，这也是容易引起错误的因素，因此脚本出错时，应该注意检查双引号、单引号、倒引号及引用内容的完整性。除此之外，还要检查以下情况。

- 变量引用符是最容易忘记的引用符之一。
- 在同一个语句中，如果需要使用单、双引号嵌套，不能只使用其中的一种，这在脚本中是不被允许的。

3. 条件测试

条件测试也是一个容易出现错误的地方。输入条件测试时，应该注意各要素和比较操作符之间应该使用空格隔开。其标准格式如下：

```
[ -w test.tmp ]
```

除此之外，条件测试中的数值测试不能使用“<”和“>”表示小于和大于，因为这两个符号已经被用于重定向了。

4. 循环错误

循环也是一个非常容易出现错误的地方，可能出现的错误如下。

- 使用了一个不适当的条件测试，导致没有循环或循环无限制。
- 错误地理解了 until 语句（条件测试为真跳出循环）和 while 语句（条件测试为假跳出循环）的工件原理。
- 没有为无限循环语句设置适当的出口。
- 使用无限循环时，应该使用 sleep 命令控制循环执行的速度，以免消耗过多的系统资源。

相比较之下，循环的错误比较容易排除，注意循环的条件测试和出口控制即可。

5．Vim编辑器

使用编辑器自带的工具，可以减少许多麻烦，例如语法高亮等。在 Linux 系统中，有许多可用于脚本编程的编辑器，RHEL5.3 默认安装的 Vim 就是其中一个。

使用 Vim 编辑器打开脚本：

```
vim filename
```

默认情况下，Vim 会使用不同的颜色显示不同的关键字，如图 19.1 所示。

使用 Vim 编辑器的语法高亮功能，可以轻易地发现脚本中的错误。

```
#!/bin/bash
BACKUP_DIR=/file/iso/backup
BACKUP_DIR_1=/file/hdb/
LOG_FILE=/root/exam_backup.log
YEAR=`date +%Y`
MOUNTH=`date +%m`
DAY=`date +%d`
HOUR=`date +%k`
MINUTE=`date +%M`
FILE_NAME=$YEAR-$MOUNTH-$DAY-$HOUR:$MINUTE.tgz
echo "----------------------------Begin---------------------------------">>
$LOG_FILE
echo "$YEAR-$MOUNTH-$DAY-$HOUR:$MINUTE:">>$LOG_FILE
find /file/exam -type f -mtime -1 -print>$BACKUP_DIR/backup_filelist 2>>$LOG_FIL
E
if [ $? -ne 0 ]
   then
      echo "find error">>$LOG_FILE
      echo "################################Error exit############################
#####">>$LOG_FILE
      exit 1
fi
tar -czT $BACKUP_DIR/backup_filelist -f $BACKUP_DIR/$FILE_NAME 2>>$LOG_FILE
if [ $? -ne 0 ]
"backup_file.sh" 29L, 1002C                                          1,1          Top
```

图 19.1　Vim 编辑器的语法高亮功能

19.3.2　脚本调试

对于任何程序编写者而言，调试都是一项非常艰巨而烦人的任务，脚本调试也不例外，特别是调试运行过程中没有提示错误、但结果不正确的脚本时。这种情况通常称为逻辑错误。导致这种现象的情况有许多，例如变量输入错误、函数之间没有衔接好等。本小节将简单介绍脚本调试的方法及使用的工具等内容。

【排除错误和调试的区别】

错误通常是指人为的语法错误，例如输入的命令错误、命令格式无法识别等。由于 Shell 无法识别，因此这类错误通常比较容易解决。调试通常是解决脚本中、程序中的逻辑错误，例如错误地将加号输入为减号，输入命令时，用错误选项等。这类错误通常称为逻辑错误。由于 Shell 不会提示逻辑错误，因此这类错误解决起来非常麻烦。

1．echo命令

调试脚本时，使用 echo 命令是一个比较笨的办法，但这个办法却非常管用，特别是在使用了大量的流程控制语句的脚本中。

调试脚本时，可以在脚本中的流程控制语句的转折点使用命令 echo 加上标记，以判断脚本执行时的流程。当然也可以使用 echo 命令判断变量的值是否正确。通过查看变量、函数返回值等，可以判断这些值是否与预期值相等，脚本执行的流程是否按预期的计划进行等。

2．退出状态

有一些函数和脚本，由于功能较为复杂，执行过程中可能会从多个出口返回、退出。因此可以人为地在这些函数和脚本的出口处，指定返回和退出状态，然后通过检测返回、退出状态的方法查看函数、脚本使用的出口。由此可以推断出函数或脚本执行时，其内部函数和变量值的变化情况。

3．set命令

调试脚本时，可能会希望使用更详细的调试工具。这时可以使用 set 命令的 x 选项，显示所有命令执行及变量值的变化过程等。为了演示 set 命令的使用，为此引入一个示例脚本 example19.2.sh。在这个简单的示例脚本中，首先使用 set -x 开启调试模式，然后再使用 while 循环读取文件 students 中的内容并输出，最后使用命令 set +x 关闭调试模式。

查看示例脚本的内容如下：

```
# cat example19.2.sh
#!/bin/bash

#this is a example script
#6/27/11

#使用 set 命令的选项 x，启动调试模式
set -x

#使用循环处理 read 命令读取文件 students 的内容
#read 命令每次读取一行，只有将文件读取完循环才会结束
while read LINE
  do
   #读取文件后，使用 echo 命令输出文件内容
   echo $LINE
done <students

#使用+x 表示关闭调试模式
set +x
```

执行这个示例脚本如下：

```
# ./example19.2.sh
#下面这行表示执行 read LINE 命令
+ read LINE
#下面这行表示输出行
+ echo 2821020225 Liulu Sichuan Lixia 01/23/93 89 76 88 72 325 81
#下面这行表示正常输出
2821020225 Liulu Sichuan Lixia 01/23/93 89 76 88 72 325 81
+ read LINE
```

```
......
+ echo 2721030227 wangtao Yunnan Huli 03/21/93 87 76 69 88 320 80
2721030227 wangtao Yunnan Huli 03/21/93 87 76 69 88 320 80
#由于读取到最后一行之后，因此下面这个命令将会执行失败
+ read LINE
#使用选项+x 表示关闭调试模式
+ set +x
```

在上面的结果中需要注意的是，使用加号“+”开头的行都是执行的命令，行首没有“+”的都是脚本输出，并且脚本中的所有变量都使用值的形式替代。

如果脚本中存在逻辑错误，在上面的调试过程中，可以清晰地看到脚本执行的流程和出错的地方。

在使用 set 命令的 x 选项启动调试模式时，不一定非要将所有的脚本语句都进行调试。如果需要，也可以使用 set -x、set +x 调试一段或多段可能存在问题的脚本。

注意：使用 set 命令开启调试模式时使用的是-x，而在关闭调试模式时使用的是+x，一定要注意“-”和“+”号的区别。

19.4　Shell 编程实例——自动备份文件脚本

在前面的数据备份章节中，介绍了备份的重要性、如何建立备份等内容。其中介绍了目前大多数互联网中的服务器都会选择在晚上执行备份。虽然晚上备份可以避免很多麻烦，但大多数管理员却都想半夜从床上爬起来，守着服务器备份。这时管理员可以自己编写一个备份文件的脚本，让系统自动备份。在本节中将介绍如何编写脚本并实现自动备份。

19.4.1　需求和设计思路

在开始编写脚本之前，明确需求是一个重要工作。在本例中需求如下。

- 如果是星期天，需要使用完全备份，其他时间都使用差异备份。
- 备份完成后，需要存放两个备份，第 1 个放在本地服务器的文件系统中，第 2 个备份要放到远程服务器上。
- 备份过程中，需要使用详细的日志，以便管理员查阅。
- 备份过程需要自动进行。

除此之外，还需要为备份文件取一个非常有意义的名字等。明确需求之后，管理员就可以针对需求着手设计脚本。

- 针对不同时间使用不同备份类型的情况，可以使用 date 命令获取当前时间，并根据不同的时间，调用不同函数实现备份。
- 针对需要保存到远程服务器的情况，可以使用挂载 NFS（Network File System，网络文件系统，是类 UNIX 系统中比较流行的文件共享服务）文件系统的方法实现，也可以使用挂载 Windows 共享的方法。
- 针对日志问题，使用单独的日志文件，以便于管理员查看。

❑ 针对自动备份的问题，可以使用 cron 计划任务的方式实现。

解决需求和设计思路之后，就可以按以上思路开始编写脚本了。

🔔**提示**：初学者编写复杂的脚本时，建议使用以上方式列出脚本的需求和解决思路，然后一步一步编写。

19.4.2　完全备份模块

在这个示例中，首先需要定义最基本的模块，完全备份函数和差异备份函数。但在这之前，应该先编写脚本的主体部分，在主体部分中定义脚本需要使用的变量。

（1）在本例中的定义部分内容如下：

```
#!/bin/bash
#This script for the backup /file directory.
#Script author:WangGang
#3/12/09
#这两个目录用于存放不同的磁盘备份
#这个脚本中没有使用变量“BACKUP FILE DIR 2”
#Backup files directory
BACKUP FILE DIR 1="/mnt/backup/file"
BACKUP FILE DIR 2="/mnt/backup file"

#定义存放备份文件名称的变量
#backup file name
BACKUP FILE NAME=null

#定义存放日志的文件
#Log file directory
LOG FILE="/root/file backup.log"

#定义相应的变量，分别存放当前年、月、日、时、分和周
#可以使用这些变量拓展脚本的功能
#date
YEAR='date +%Y'
MOUNTH='date +%m'
DAY='date +%d'
HOUR='date +%k'
MINUTE='date +%M'
WEEK='date +%w'
```

完成最基本的变量定义之后，就可以开始编写完全备份函数。

（2）完全备份函数需要使用 tar 命令备份参数指定的目录，并根据 tar 命令的退出状态设置函数的返回状态。

在本例中完全备份函数的内容如下：

```
#定义完全备份函数 backup_full
#函数使用了两个参数，分别是要备份的目录、存放备份文件的路径和名称
#Backup type:full
function backup_full()
```

```
{
   #保存当前目录到变量 DIR 中
   DIR='pwd'
   #使用命令 dirname 获取目标目录的上级目录
    DIR='dirname $1'
   #获取要备份的目录名称
   BACKUP DIR='basename $1'

   #进入备份目录的上级目录开始备份
   cd $ DIR
   tar -czf $2 $BACKUP DIR &>/dev/null

   #如果备份成功，则返回原目录、清除变量并返回
   if [ $? = 0 ]
     then
       cd $DIR
       unset DIR  DIR BACKUP DIR
       return 0
   else
       #如果备份失败，则返回原目录、清除使用的变量，返回函数状态为失败
       cd $DIR
       unset DIR  DIR BACKUP DIR
       return 1
   fi
}
```

当然除了定义完全备份函数之外，还需要编写主体部分来测试完全备份函数，以确保完全备份函数的正确。

19.4.3　差异备份模块

完成完全备份模块的定义之后，就可以开始第二项工作定义差异备份模块。在差异备份模块中，需要先使用 find 命令和参数查找需要备份的文件，并保存在临时文件中。最后再使用 tar 命令备份临时文件中列出的文件，并根据命令的退出状态设置函数的返回状态。

在本例中差异备份模块的内容如下：

```
#定义差异备份使用的函数 backup incremental
#函数使用 3 个参数，分别是要备份的目录、时间、存放备份文件的路径和名称
#时间参数使用的单位为天
#backup type:incremental
function backup incremental()
{
   #保存当前目录到变量 DIR 中
   DIR='pwd'
   #保存目标目录的上级目录
    DIR='dirname $1'
   #保存要备份的目录名称
   BACKUP DIR='basename $1'

   #进入需要备份目录的上级目录
```

```
  cd $ DIR
  #使用 find 命令查找需要备份的文件，这些文件满足两个条件
  #第 1，文件类型为普通文件
  #第 2，文件的修改时间戳记在指定的时间以内
  #将找到的文件放入临时文件/tmp/backup_file_list 中
  find ./$BACKUP DIR -type f -mtime -$2 -print >/tmp/backup file list
  2>/dev/null

  if [ $? = 0 ]
    then
      #如果 find 命令执行成功，则使用 tar 命令备份找到的文件
      tar -czT /tmp/backup file list -f $3 &>/dev/null
      #如果 tar 命令备份文件成功，则清除变量、删除临时文件并返回
      if [ $? = 0 ]
        then
          unset DIR  DIR BACKUP DIR
          rm -rf /tmp/backup file list &>/dev/null
          return 0
      #tar 命令执行失败时，清除变量、删除临时文件并返回
      else
          unset DIR  DIR BACKUP DIR
          rm -rf /tmp/backup file list &>/dev/null
          return 1
      fi
  else
      #find 命令执行失败时，清除变量、删除可能的临时文件，然后返回
      unset DIR  DIR BACKUP DIR
      rm -rf /tmp/backup file list &>/dev/null
      return 1
  fi
}
```

与编写的完全备份函数一样，编写差异备份时，也需要编写主体部分以测试差异备份函数。

注意：由于使用时间戳记作为文件更新标记，因此函数中实现的是差异备份，还是增量备份都依赖于函数收到的时间参数。鉴于此，在本例中将其称为差异备份。

19.4.4　远程备份模块

定义备份模块之后，需要定义远程备份模块。在本例中远程备份操作的流程是先挂载远程文件系统，然后将备份文件复制到远程文件系统中，最后卸载远程文件系统。

实现这个远程备份功能的模块内容如下：

```
#定义用于上传备份文件的函数 backup_nfs
#函数使用远程服务器的 nfs 目标作为目标目录，将备份文件上传至远程服务器
#函数使用备份文件的名称作为参数
#function backup nfs
#upload backup file
function backup_nfs()
```

```
{
  #使用 mount 命令将远程服务器的 nfs 目录挂载到本地的/mnt/nfs 目录中
  #此处假定本地用于挂载的目录已经存在
  mount -t nfs 192.168.118.226:/mnt/backup /mnt/nfs &>/dev/null
  if [ $? = 0 ]
    then
      #如果挂载命令执行成功，则将参数指定的文件使用 cp 命令上传至远程服务器
      if 'cp $1 /mnt/nfs/'
        then
          #如果挂载和复制都执行成功了，则输出提示到日志文件、卸载远程文件系统并返回
          echo "Upload successful." >>$LOG_FILE
          umount /mnt/nfs
          return 0
      else
          #如果上传失败，则输出提示到日志文件、卸载远程文件系统并返回
          echo "Upload failed." >>$LOG_FILE
          umount /mnt/nfs
          return 1
      fi
  else
      #如果挂载远程文件系统失败，则输出提示到日志文件并返回
      echo "mount nfs failed." >>$LOG_FILE
      return 1
  fi
}
```

在本例中使用 NFS 作为远程文件系统，如果初学者不熟悉其使用方法，也可以使用 Windows 系统中的共享、FTP 服务器等。

19.4.5　主体和日志功能

在本例中，先判断脚本运行时间是否为星期天，如果是星期天，就调用完全备份函数，否则就调用差异备份函数。在整个主体结构中，按函数的执行状态设置日志。需要注意的是在主体结构中，并没有单独的日志模块，也没有使用 logger 命令将日志写入系统日志中，而是在脚本主体结构中，直接将日志写入日志文件中。

脚本主体结构内容如下：

```
#判断当前是否为星期天，如果是则使用完全备份
#if it is sunday
#use function backup_full
if [ $WEEK = 0 ]
  then
    #定义备份之后的备份文件名称并保存在 BACKUP_FILE_NAME 变量中
    BACKUP_FILE_NAME="$BACKUP_FILE_DIR_1/$YEAR-$MOUNTH-$DAY-$HOUR-
    $MINUTE.full.tar.gz"

    #调用 backup_full 函数对目录/file/soft 执行完全备份
    backup_full "/file/soft" "$BACKUP_FILE_NAME"

    #判断 backup_full 函数是否执行失败
    if [ $? != 0 ]
```

```
    then
      #如果完全备份失败，就向日志文件写入备份开始信息
      echo "###################Backup begin######################"
      >>$LOG FILE
      #将当前时间写入日志文件
      echo "time:$YEAR-$MOUNTH-$DAY $HOUR:$MINUTE" >>$LOG FILE
      #向日志文件写入备份错误及备份结束信息
      echo "backup error."  >>$LOG FILE
      echo "###################End###############################"
      >>$LOG FILE
      #清除所有使用过的变量、函数并设置退出状态
      unset BACKUP FILE DIR 1 BACKUP FILE DIR 2 BACKUP FILE NAME LOG FILE
      unset YEAR MOUNTH DAY HOUR MINUTE WEEK
      unset backup full backup incremental backup nfs
      exit 1
  fi

  #如果 backup_full 函数执行成功，就向日志文件写入备份开始、备份时间、备份成功信息
  echo "###################Backup begin######################"
  >>$LOG FILE
  echo "time:$YEAR-$MOUNTH-$DAY $HOUR:$MINUTE" >>$LOG FILE
  echo "backup success."  >>$LOG FILE
  #调用 backup_nfs 函数将备份文件上传至远程服务器
  backup nfs $BACKUP FILE NAME

  #如果上传备份文件函数执行失败
  #就输出结束信息至日志文件、清除使用过的变量函数并设置退出状态
  if [ $? != 0 ]
    then
      echo "###################End###############################"
      >>$LOG FILE
      unset BACKUP FILE DIR 1 BACKUP FILE DIR 2 BACKUP FILE NAME LOG FILE
      unset YEAR MOUNTH DAY HOUR MINUTE WEEK
      unset backup full backup incremental backup nfs
      exit 1
  fi

  #如果上传成功，则输出结束信息至日志文件、清除变量函数并设置退出状态
  echo "###################End###############################"
  >>$LOG FILE
  unset BACKUP FILE DIR 1 BACKUP FILE DIR 2 BACKUP FILE NAME LOG FILE
  unset YEAR MOUNTH DAY HOUR MINUTE WEEK
  unset backup full backup incremental backup nfs
  exit 0
fi

#如果不是星期天，就使用差异备份
#if not sunday
#use function backup incremental
#将备份后的文件名称保存至变量 BACKUP FILE NAME 中
BACKUP FILE NAME="$BACKUP FILE DIR 1/$YEAR-$MOUNTH-$DAY-$HOUR-$MINUTE.i
ncremental.tar.gz"
```

```
#调用 backup_incremental 函数对目录/file/soft 执行差异备份，时间参数设置为 1 天
backup incremental "/file/soft" "1" "$BACKUP FILE NAME"

#判断 backup_incremental 函数执行是否失败
if [ $? != 0 ]
  then
    #如果执行失败，就将开始信息、时间、错误信息和结束信息写入日志文件
    echo "###################Backup begin######################"
    >>$LOG FILE
    echo "time:$YEAR-$MOUNTH-$DAY $HOUR:$MINUTE" >>$LOG_FILE
    echo "backup error."  >>$LOG FILE
    echo "###################End###############################"
    >>$LOG FILE
    #清除使用过的变量、函数并设置退出状态
    unset BACKUP FILE DIR 1 BACKUP FILE DIR 2 BACKUP FILE NAME LOG FILE
    unset YEAR MOUNTH DAY HOUR MINUTE WEEK
    unset backup full backup incremental backup nfs
    exit 1
fi

#如果执行成功，则调用函数 backup_nfs 将备份文件上传至远程服务器
backup_nfs $BACKUP_FILE_NAME

#如果函数执行失败，就将信息写入日志文件、清除变量函数并设置退出状态
if [ $? != 0 ]
  then
    echo "###################End###############################"
    >>$LOG FILE
    unset BACKUP FILE DIR 1 BACKUP FILE DIR 2 BACKUP FILE NAME LOG FILE
    unset YEAR MOUNTH DAY HOUR MINUTE WEEK
    unset backup full backup incremental backup nfs
    exit 1
fi

#如果上传函数执行成功，就将结束信息写入日志文件、清除使用过的变量并设置退出状态
echo "###################End###############################" >>$LOG FILE
unset BACKUP FILE DIR 1 BACKUP FILE DIR 2 BACKUP FILE NAME LOG FILE
unset YEAR MOUNTH DAY HOUR MINUTE WEEK
unset backup_full backup_incremental backup_nfs
exit 0
```

在上面的主体结构中，差异备份的时间为 1 天，读者也可以修改这个时间，以实现更灵活的备份策略。

提示： 初学者可以通过改进这个脚本学习如何编写系统管理脚本。例如完善的方向有：使用单独的日志文件并通过日志守护进程 syslogd 收集日志，为日志功能添加单独的模块，简化脚本结构等。

19.4.6　自动运行备份脚本

为了实现自动备份功能，还需要将备份脚本加入到计划任务中。运行命令 crontab –e 打开计划任务编辑窗口，然后在其中输入以下内容：

```
* 3 * * * /root/backup_file.sh
```

保存之后，系统就会在每天凌晨 3 点开始备份数据了。

19.5　Shell 编程实例——防火墙快速配置脚本

如果管理员需要安装多台服务器，配置防火墙是一个很麻烦的工作，这是因为配置防火墙可能需要输入许多命令。本节将简单介绍一个用于快速配置防火墙的脚本。

19.5.1　设置防火墙状态

Linux 系统通常都自带有一个名为 iptables 的防火墙（较早的版本可能使用的是 ipchains），这个防火墙虽然功能强大，但配置起来却也非常麻烦。在本例中将编写一个脚本实现快速配置 iptables 防火墙。

防火墙的设置语句并不复杂，通常开启一个端口或协议只需要使用 1-2 行语句即可，因此在这个脚本中，没有使用过多的函数。

（1）编写防火墙配置脚本的首要任务仍然是先定义脚本使用的变量：

```
#!/bin/bash
#This is a fast firewall configuration script.
#Script author:
#3/8/09
#设置系统防火墙状态变量
IPTABLES_STATUS=0
#设置 OUTPUT 和 INPUT 链的默认规则变量
DEFAULT_OUTPUT=0
DEFAULT_INPUT=0

#设置系统服务 SSH 端口规则变量
#该端口主要用于远程管理
RULE_SSH=0
#设置 ICMP 协议规则变量
#该规则将决定外界是否能使用 ping 命令测试与主机的连通性
RULE_ICMP=0

#设置 INPUT 链需要打开的端口列表
OPEN_IN_PORT=null
#设置 OUTPUT 链需要打开的端口列表
OPEN_OUT_PORT=null
```

```
#设置命令头变量
IPT="/sbin/iptables"
```

提示：在 iptables 防火墙中，规则一般都放在链中。INPUT 和 OUTPUT 链主要用于存放主机接收、发送数据包的规则。

（2）定义好需要使用的变量之后，就可以开始编写设置防火墙状态的函数和语句了。在设置防火墙状态函数中，按参数的值设置防火墙的状态。如果参数为 0，就开启防火墙，否则就关闭防火墙。

设置防火墙状态函数的内容如下：

```
#设置系统防火墙状态的函数 iptables_status
#该函数使用 1 个参数，如果等于 0，则表示开启防火墙，否则就关闭防火墙
#function iptables status
#Used to set the firewall status
function iptables status()
{
   #判断参数 1 是否等于 0
   if [ $1 = 0 ]
     then
       #如果参数 1 等于 0，就设置系统防火墙在运行级别 345 为开启状态
       #并根据命令执行的退出状态设置返回状态
       if 'chkconfig --level 345 iptables on &>/dev/null'
         then
           return 0
       else
           return 1
       fi
   else
       #如果参数不等于 0，则关闭防火墙的自启动状态
       #并根据命令执行的退出状态设置返回状态
       if 'chkconfig iptables off &>/dev/null'
         then
           return 0
       else
           return 1
       fi
   fi
}
```

（3）编写以上函数之后，还需要在主体结构中编写与用户交互的内容。在主体结构中，使用 while 和 case 语句要求用户输入是否开启防火墙。

查看主体结构的内容如下：

```
#判断当前的执行者是否为 root 用户，如果不是就退出脚本
#if the user is not root
if [ "'whoami'" != "root" ]
  then
    echo "You need to be root to run this script" >2
    exit
```

```
fi

while true
  do
    #提示是否要开启系统防火墙并接收用户输入
    echo -n "Open the system firewall?[yes|no]"
    read ANS
    case "$ANS" in
      #如果用户输入的是 y、Y、yes 或 Yes，就设置变量并跳出
      y|Y|yes|Yes)
        echo "please wait..."
        IPTABLES STATUS=0
        break
        ;;
      #如果用户输入的是 n、N、no 或 No，就设置变量并跳出
      n|N|no|No)
        echo "please wait..."
        IPTABLES STATUS=1
        break
        ;;
      #当用户输入不符合要求时，提示用户输入指定的值
      #并开始下一个循环要求用户重新输入
      *)
        echo "Please enter yes or no."
        continue
        ;;
    esac
done

#调用 iptables_status 函数设置防火墙状态
echo ""
iptables status $IPTABLES STATUS

#根据 iptables_status 函数返回状态输入提示信息
if [ $? = 0 ]
  then
    echo "Set system firewall status success."
else
    echo "Set system firewall status faild."
Fi
```

由于设置防火墙是管理员的工作之一，因此在脚本主体部分首先判断当前运行脚本的用户是否为 root 用户。如果不是 root 用户，就输出提示信息并退出。

19.5.2　添加防火墙规则

设置完防火墙状态之后，就可以根据防火墙的状态（防火墙关闭时无法为其设置状态），为防火墙设置规则了。

（1）在本例中，需要设置 INPUT 和 OUTPUT 链的默认规则，自定义的规则还有 SSH 服务端口、ICMP 协议，以及用户在 INPUT 和 OUTPUT 链上的自定义端口等。

添加防火墙规则过程中与用户交互的脚本内容如下：

```
#判断并设置防火墙状态
#判断变量 IPTABLES_STATUS 是否等于 1
if [ $IPTABLES_STATUS = 1 ]
  then
    #如果防火墙状态变量 IPTABLES_STATUS 等于 1，则关闭系统防火墙并输出提示
    echo "Shut down system firewall."
    echo "Please wait..."
    /etc/init.d/iptables stop &>/dev/null
    echo "OK!"
    exit 0
else
    #如果防火墙状态变量值不等于 1，则开启系统防火墙并输出提示
    echo "Start system firewall."
    echo "Please wait..."
    /etc/init.d/iptables start &>/dev/null
    echo "OK!"
fi

echo ""
while true
  do
    #提示用户输入系统防火墙 INPUT 链的默认规则并接收用户输入
    echo -n "Please enter a default rule INPUT chains[ACCEPT|DROP]:"
    read ANS
    case $ANS in
      #如果用户输入 ACCEPT、accept 或 a 时
      #设置变量 DEFAULT_INPUT 的值等于 0 并跳出
      ACCEPT|accept|a)
              DEFAULT_INPUT=0
              break
              ;;
      #如果用户输入的是 DROP、drop 或 d 时
      #设置变量 DEFAULT_INPUT 的值为 1 并跳出
      DROP|drop|d)
              DEFAULT_INPUT=1
              break
              ;;
      *)
              #当用户输入不符合要求时，提示用户并返回要求用户重新输入
              echo "Please enter ACCEPT or DROP."
              continue
              ;;
    esac
done

while true
  do
    #提示用户输入系统防火墙 OUTPUT 链的默认规则并接收用户输入
    echo -n "Please enter a default rule OUTPUT chains[ACCEPT|DROP]:"
    read ANS
```

```
    case $ANS in
      #如果用户输入 ACCEPT、accept 或 a
      #设置变量 DEFAULT_OUTPUT 的值等于 0 并跳出
      ACCEPT|accept|a)
              DEFAULT OUTPUT=0
              break
              ;;
      #如果用户输入的是 DROP、drop 或 d
      #设置变量 DEFAULT_OUTPUT 的值为 1 并跳出
      DROP|drop|d)
              DEFAULT OUTPUT=1
              break
              ;;
      *)
              #当用户输入不符合要求时，提示用户并返回要求用户重新输入
              echo "Please enter ACCEPT or DROP."
              continue
              ;;
    esac
done

echo ""
while true
  do
    #提示用户是否需要开启服务 SSH 的端口并接收用户输入
    echo -n "Need to open the SSH service port?[yes|no]"
    read ANS
    case $ANS in
      #如果用户输入的是 y、Y、yes 或 Yes，就设置变量并跳出
      yes|Yes|y|Y)
              RULE SSH=0
              break
              ;;
      #如果用户输入的是 n、N、no 或 No，就设置变量并跳出
      no|No|n|N)
              RULE SSH=1
              break
              ;;
      *)
              #当用户输入不符合要求时，提示用户并进入下一个循环要求用户重新输入
              echo "Please enter yes or no."
              continue
              ;;
    esac
done

echo ""
while true
  do
    #提示用户是否需要开启 ICMP 协议并接收用户输入
    echo -n "Need to open the ICMP protocol?[yes|no]"
    read ANS
```

```
    case $ANS in
      #根据用户输入值设置相应的变量
      yes|Yes|y|Y)
              RULE_ICMP=0
              break
              ;;
      no|No|n|N)
              RULE_ICMP=1
              break
              ;;
      *)
              echo "Please enter yes or no."
              continue
              ;;
    esac
done

#提示用户输入在 INPUT 链上开启的端口并接收输入
echo ""
echo -n "please enter the port open on the INPUT chains(eg:80,53…):"
read OPEN_IN_PORT
#提示用户输入需要在 OUTPUT 链上开启的端口并接收用户输入
echo -n "please enter the port open on the OUTPUT chains(eg:80,53…):"
read OPEN_OUT_PORT
```

（2）取得用户的设置之后，就可以利用上面的变量值添加防火墙的规则了。在本例中，添加防火墙规则的脚本内容如下：

```
#输出提示信息，并清除防火墙的规则
echo ""
echo "Begin setting the system firewall..."
echo -n "Clear filewall rules..."
$IPT -F
$IPT -X
#$IPT -t nat -F
#$IPT -t nat -X
echo "OK!"

#提示用户设置防火墙的默认规则
echo ""
echo -n "Setting the system firewall default rules..."
#根据变量 DEFAULT_INPUT 的值设置 INPUT 链上的默认规则
if [ $DEFAULT_INPUT = 0 ]
  then
    #如果变量值等于 0，就将 INPUT 链上的默认规则设置为接受
    $IPT -P INPUT ACCEPT
else
    #如果变量值不等于 0，就丢弃 INPUT 链上的所有数据包
    $IPT -P INPUT DROP
fi

#根据变量 DEFAULT_OUTPUT 的值设置 OUTPUT 链上的默认规则
```

```
if [ $DEFAULT_OUTPUT = 0 ]
  then
    $IPT -P OUTPUT ACCEPT
else
    $IPT -P OUTPUT DROP
fi
echo "OK!"

echo ""
echo -n "Setting system service rules..."
#根据变量 RULE_SSH 的值设置 SSH 服务端口的状态
if [ $RULE_SSH = 0 ]
  then
    #如果变量 RULE_SSH 的值等于 0，就在 INPUT 和 OUTPUT 链上开启端口
    $IPT -t filter -A INPUT -p tcp --dport 22 -j ACCEPT
    $IPT -t filter -A OUTPUT -p tcp --sport 22 -j ACCEPT
else
    #如果变量 RULE_SSH 的值不等于 0，就丢弃 22 端口的数据包
    $IPT -t filter -A INPUT -p tcp --dport 22 -j DROP
    $IPT -t filter -A OUTPUT -p tcp --sport 22 -j DROP
fi

#根据变量 RULE_ICMP 的值设置 ICMP 协议状态
if [ $RULE_ICMP = 0 ]
  then
    $IPT -t filter -A INPUT -p icmp -j ACCEPT
    $IPT -t filter -A OUTPUT -p icmp -j ACCEPT
else
    $IPT -t filter -A INPUT -p icmp -j DROP
    $IPT -t filter -A OUTPUT -p icmp -j DROP
fi
echo "OK!"

#设置用户自定义的端口在防火墙上的状态
echo ""
echo -n "Add custom rules..."
$IPT -t filter -A INPUT -p tcp -m multiport --sport $OPEN_IN_PORT -j ACCEPT
$IPT -t filter -A INPUT -p udp -m multiport --sport $OPEN_IN_PORT -j ACCEPT

$IPT -t filter -A OUTPUT -p tcp -m multiport --dport $OPEN_OUT_PORT -j ACCEPT
$IPT -t filter -A OUTPUT -p udp -m multiport --dport $OPEN_OUT_PORT -j ACCEPT
echo "OK!"
```

到此添加防火墙规则的脚本就已经编写完成了。

19.5.3　保存防火墙规则

在前面两个小节中，已经编写了设置防火墙状态、添加规则的脚本内容。接下来就需要运用 iptables 服务保存添加的防火墙规则，以免服务器重启后规则丢失。

在本例中保存防火墙规则脚本内容如下：

```
#提示用户并保存生成的防火墙规则
echo ""
echo "Save the rules to the configuration file..."
echo -n "please wait..."
/etc/init.d/iptables save &>/dev/null
echo "OK!"

#清除变量和函数并退出
unset IPTABLES STATUS DEFAULT OUTPUT DEFAULT INPUT RULE SSH RULE ICMP
unset OPEN IN PORT OPEN OUT PORT IPT iptables status
echo "Set complete."
exit 0
```

由于防火墙是一个比较专业的内容，因此初学者可能需要阅读额外的资料了解这个脚本中使用的 iptables 命令。初学者也可以通过修改、完善这个脚本的方式学习脚本编程。修改和完善方向有：将脚本进一步模块化、使用步进循环处理自定义端口列表等。

19.6　快速初始化系统脚本

通常管理员安装完操作系统后，会立即设置操作系统。如果管理员需要安装的操作系统有许多，重复设置操作系统将会是一项非常繁琐的工作。在本节中将介绍如何编写一个脚本快速初始化系统。

19.6.1　初始化系统网络

在无人值守安装的操作系统中，通常都没有设置网络。因此快速初始化系统脚本的第 1 件事，就是初始化系统网络。

1．定义脚本主体和变量

在初始化系统网络之前，先编写脚本主体部分并定义脚本中使用的变量：

```
#!/bin/bash
#This is a system initialization script.
#author:
#1/23/09

#设置保存原始配置文件的目录
BACKUP CONFIG DIR=/root/backup config

#设置变量保存当前目录及网络接口初始化目录
DIR='pwd'
NET DIR="/etc/sysconfig/network-scripts"

#设置用于保存网络接口、ip 地址、子网掩码及网关的变量
INTER=null
IP_ADDR=null
```

```
NET MASK=null
NET GW=null
#设置网关配置标记
NET GW TRUE=0

#设置变量 DNS_PRI 和 DNS_SEC，用于保存主要 DNS 和次要 DNS 的地址
DNS PRI=null
DNS_SEC=null
```

定义完上述信息之后，就可以开始编写各种函数了。

2．定义初始化网络的函数

初始化网络需要使用的函数有许多，例如检测接口函数、接收用户输入的 IP 地址函数、DNS 配置函数等。此处将一一介绍这些函数。

（1）首先需要定义一个函数，用于接收和测试用户输入的网络接口。在这个函数中，使用命令 ifup、ifdown 判断用户输入的网络接口是否正确。

函数的内容如下：

```
#定义用于接收和测试用户输入的网络接口函数 input_interface
#function input interface
function input interface()
{
   #输出提示并要求用户输入要设置的网络接口
   echo -n "Please enter network interface:"
   read INTER
   #输出提示信息并使用 ifup 和 ifdown 命令检测用户的输入是否正确
   echo "Testing,please wait..."
   ifup $INTER &>/dev/null
   ifdown $INTER &>/dev/null
   #使用上一条命令的退出状态作为函数的返回状态
   return $?
}
```

与前面的脚本不同，此处使用命令测试用户的输入是否正确。

（2）接下来需要定义用于接收用户输入 IP 地址、子网掩码的函数：

```
#设置处理用户输入 IP 地址的函数
#function input ip address
function input ip addr()
{
   #输出提示并要求用户输入 IP 地址和子网掩码
   echo -n "Please enter interface $INTER use IP address(eg:192.168.0.2):"
   read IP ADDR
   echo -n "Please enter net mask(eg:255.255.255.0):"
   read NET MASK
   echo "Testing,please wait..."
   #使用 ifconfig 命令判断用户输入是否正确
   ifconfig $INTER $IP ADDR netmask $NET MASK up &>/dev/null
   #使用 ifconfig 命令的退出状态作为函数的返回状态
   return $?
```

```
}
```

在这个函数中，使用 ifconfig 命令来测试 IP 地址和子网掩码是否正确，这样做的目的是为了简化脚本。因为如果使用正则表达式测试用户的输入是否为合法的 IP 地址、子网掩码可能会非常麻烦。

（3）接收了用户输入的 IP 地址和子网掩码之后，需要定义接收默认网关的函数：

```
#设置默认网关输入函数
#function input gateway ip address
function input gw()
{
   #提示用户输入网关并接收用户输入
   echo -n "Please enter default gateway(eg:192.168.0.1):"
   read NET GW
   #使用 route 命令测试输入的网关地址是否正确
   route add default gw $NET GW &>/dev/null
   #使用上一条件命令的退出状态设置函数返回状态
   return $?
}
```

与前面的几个函数一样，这个函数也使用了 route 命令测试用户输入的默认网关是否为一个合法的地址。

（4）读者可能注意到：虽然用户可能输入了一个正确的默认网关地址，但如何确保默认网关是正确的？这时可以通过网络连通性测试来判断。

定义连通性测试函数如下：

```
#设置用于测试连通性的函数
#此函数的参数只有一个，即目标主机的 IP 地址
#function test connection
function test conn()
{
   echo "Testing,please wait..."
   #使用 ping 命令测试与参数地址是否能通信
   ping $1 -c 2 &>/dev/null
   return $?
}
```

（5）如果网关测试未能通过，这时就需要一个用于处理网关错误的函数：

```
#定义用于处理默认网关输入错误的处理函数
#function input gateway error
function input_gw_error()
{
       #提示用户输入的默认网关错误并询问用户需要重新输入或保存已有配置
       echo "Input default gateway error." >2
       echo -n "re-enter or save current configure?[enter|save]"
       read ANS
       case $ANS in
         enter|Enter)
           #当用户输入 enter 时，跳转并提示用户重新输入默认网关
           continue
```

```
            ;;
          save|Save)
            #如果用户为 save 时调用函数保存网络接口信息
            save inter ip $INTER $IP ADDR $NET MASK
            #设置网关状态为 1
            NET GW TRUE=1
            echo "OK!"
            return 0
            ;;
          *)
            #用户输入错误时重新要求用户输入默认网关
            echo "Input error,re-enter." >2
            continue
            ;;
        esac
}
```

可能读者注意到了，在上面的函数中调用了另一个用于保存网络接口配置信息的函数。不错，接下来就应该定义保存网络接口信息函数了。

（6）当默认网关测试通过，或用户需要执意保存网络接口信息时，就需要调用保存网络接口信息的函数了。保存配置文件函数使用重定向的方式写入内容如下：

```
#定义保存网络接口配置文件函数
#function save interface ip address
function save inter ip()
{
   #备份网络接口配置文件
   cp $NET DIR/ifcfg-$1 $BACKUP CONFIG DIR/ifcfg-$1.'date +"%Y-%m-%d.
   %H-%M"'
   echo "Writing configuration files"
   #将参数传递来的初始化信息写入接口初始化文件
   echo "TYPE=Ethernet" >$NET DIR/ifcfg-$1
   echo "DEVICE=$1" >>$NET DIR/ifcfg-$1
   echo "ONBOOT=yes" >>$NET DIR/ifcfg-$1
   echo "BOOTPROTO=static" >>$NET DIR/ifcfg-$1
   echo "IPADDR=$2" >>$NET DIR/ifcfg-$1
   echo "NETMASK=$3" >>$NET DIR/ifcfg-$1
   echo "GATEWAY=$4" >>$NET DIR/ifcfg-$1
   echo "Restart network inter $1,please wait..."
   #使用新初始化文件重新启动网络接口
   ifdown $1 &>/dev/null
   ifup $1 &>/dev/null
   echo "OK!"
}
```

在上面的函数中，为确保原始配置文件不被破坏，需要先备份接口配置文件，然后再保存新的配置信息，推荐在所有配置文件中使用这种做法。

（7）IP 地址、子网掩码和默认网关都正确配置之后，还应该配置 DNS 服务器，测试 DNS 服务器的方式仍然使用连通性测试。

接收用户输入的 DNS 服务器地址函数内容如下：

```
#定义用于输入 DNS 地址的函数
#function input dns ip address
function input dns()
{
   while true
     do
       #提示用户输入主 DNS 并接收用户的输入
       echo -n "Please input primary DNS:"
       read DNS PRI
       #使用函数 test_conn 检测是否能与用户输入的主 DNS 进行通信
       test conn $DNS PRI
       if [ $? = 0 ]
         then
           while true
             do
               #提示用户输入次 DNS 服务器 IP 地址并检测
               echo -n "Please input Secondary DNS:"
               read DNS SEC
               test conn $DNS SEC
               if [ $? = 0 ]
                 then
                   #如果与用户输入的次 DNS 也能通信则跳出
                   break 2
               else
                   #如果不能与用户输入的次 DNS 进行通信，则要求重新输入
                   echo "Input second DNS error." >2
                   continue
               fi
           done
       else
           #如果不能与主 DNS 进行通信，则要求用户重新输入
           echo "Input primary DNS error." >2
           continue
       fi
   done
}
```

在上面这个函数中，仍然使用连通性测试 DNS 服务器地址的正确性。

（8）当 DNS 地址输入正确之后，就需要保存用户输入的 DNS 地址。保存 DNS 地址函数的内容如下：

```
#定义保存用户输入的 DNS 地址的函数
#function save dns ip address
function save dns()
{
  #提示用户并对配置文件进行备份
  echo "Save DNS to configure files."
  cp /etc/resolv.conf $BACKUP CONFIG DIR/resolv.conf.'date +"%Y-%m-%d.
  %H-%S"'
  #将 DNS 服务器 IP 地址信息写入到配置文件
  echo "nameserver $1" >/etc/resolv.conf
  echo "nameserver $2" >>/etc/resolv.conf
```

```
  echo "OK!"
}
```

在这个函数中，仍然使用了先备份、后修改的方法将 DNS 地址写入配置文件中。

（9）并不是所有网络都需要手动输入 IP 地址、子网掩码等配置信息，有些网络会从 DHCP 服务器处获取 IP 地址。这时需要添加使用 DHCP 的配置文件函数：

```
#设置使用 DHCP 方式初始化信息的文件内容
#save interface configure file
function save inter dhcp()
{
   #使用 cp 命令备份要配置的网络接口文件
   cp $NET DIR/ifcfg-$1 $BACKUP CONFIG DIR/ifcfg-$1.'date +"%Y-%m-d.
   %H-%M"'
   #保存接口初始化信息到相应的文件中
   echo "DEVICE=$1" >$NET DIR/ifcfg-$1
   echo "ONBOOT=yes" >>$NET DIR/ifcfg-$1
   echo "BOOTPROTO=dhcp" >>$NET DIR/ifcfg-$1
   echo "TYPE=Ethernet" >>$NET DIR/ifcfg-$1
}
```

添加完以上函数之后，初始化网络的所有函数功能都已经添加完成了。

3. 主体部分

完成所有需要使用的函数编写之后，就可以整合已经编写的测试脚本形成脚本的主体部分。主体部分先判断当前用户是否为 root，然后依次要求用户输入接口名、IP 地址、子网掩码、默认网关和 DNS 服务器地址，对用户的所有输入进行测试、保存。

在本例中主体部分如下：

```
#脚本最开始处先判断当前用户是否为 root
if [ "'whoami'" != "root" ]
  then
    #如果当前用户不是 root，则输出提示并退出
    echo "You need to be root to run this script." >2
    exit 1
fi

#创建用于保存原始配置文件的目录
#mkdir backup directory
mkdir $BACKUP CONFIG DIR &>/dev/null

#使用 while 语句处理网络接口初始化
while true
  do
    #initialize network
    #输出提示并输出接口列表
    echo "initialize network..."
    cd $NET DIR
    echo "Discover network interfacenet:"
    ls ifcfg-eth* |sed 's/ifcfg-//' | grep -v "eth.."
```

```
cd $DIR

#使用 input_interface 函数处理用户输入的接口信息
#input network interface
input interface
if [  $? = 0 ]
  then
    #用户输入的接口名正确时，输出提示
    echo "OK!"
else
    #用户输入的接口名错误时，跳转并要求用户重新输入
    echo "Input interface error."
    continue
fi

#使用 while 语句处理用户输入的接口初始化的方式
#input dhcp and static
while true
  do
    echo ""
    #提示用户输入网络接口初始化方式并接收用户输入
    echo -n "Please enter the IP address for the way[dhcp|static]:"
    read ANS
    case $ANS in
      dhcp|DHCP|Dhcp)
         #如果用户输入使用 DHCP 方式初始化网络接口，输出提示并保存信息至配置文件
         echo "write configrue file."
         save inter dhcp $INTER
         echo "OK!"
         #输出提示信息后重新启动接口
         echo "Restart network interface,please wait..."
         ifdown $INTER &>/dev/null
         ifup $INTER &>/dev/null
         echo "OK!"
         #输出提示并跳出
         echo "Network configure end."
         break 2
         ;;
      static|STATIC|Static)
         #如果用户输入使用静态设置初始化网络接口则跳出
         break
         ;;
      *)
         #如果用户输入了其他值，则要求用户重新输入
         echo "Input error." >2
         continue
    esac
done

#使用 while 语句处理用户输入的 IP 地址和子网掩码
#input ip address and netmask
while true
```

```
    do
      echo ""
      #调用 input_ip_addr 函数处理用户输入的 IP 地址和子网掩码
      input ip addr
      if [ $? = 0 ]
        then
          #如果函数返回值为 0，则输出提示并跳出循环
          echo "OK!"
          break
      else
          #如果函数返回值为 1，则输出提示并跳转，要求重新输入 IP 地址和子网掩码
          echo "Input ip address or netmask error." >2
          continue
      fi
  done

  #使用 while 语句处理用户输入的默认网关
  #input default gateway ip address
  while true
    do
      echo ""
      #调用 input_gw 函数要求用户输入默认网关
      input gw
      #判断函数的返回状态
      if [ $? = 0 ]
        then
          #调用函数 test_conn 测试与默认网关是否能够连通
          test conn $NET GW
          if [ $? = 0 ]
            then
              #如果与默认网关能连通，则调用 save_inter_ip 函数
              #保存初始化信息至接口配置文件中
              save inter ip $INTER $IP ADDR $NET MASK $NET GW
              echo "OK!"
              break
          else
              #如果与用户输入的默认网关不能连接，则调用函数 input_gw_error
              #询问用户应该如何处理
              input gw error
          fi
      else
          #如果用户输入的默认网关错误
          #调用函数 input gw error 询问是否要重新输入或直接保存
          input gw error
      fi
  done

  #提示网络配置完成并跳出
  echo ""
  echo "Network interface configure end."
  break
done
```

```
#为系统添加 DNS 服务器地址
#add DNS server ip address
#添加之前检查默认网关设置标记
if [ $NET_GW_TRUE = 0 ]
  then
    #如果默认网关可用，则使用 while 语句处理用户输入的 DNS 服务器 IP 地址
    while true
      do
        echo ""
        #提示用户是否需要设置 DNS 服务并接收用户输入
        echo -n "configure DNS server?[yes|no]"
        read ANS
        case $ANS in
          yes|Yes|y|Y)
            #用户输入 yes 时调用函数 input_dns 处理 DNS 服务器地址输入
            input_dns
            echo "OK!"
            #用户输入 DNS 服务器地址后调用 save_dns 将服务器地址保存至配置文件
            save_dns $DNS_PRI $DNS_SEC
            break
            ;;
          no|No|n|N)
            #用户输入 no 时跳出 while 语句
            break
            ;;
          *)
            #用户输入错误时提示用户并要求用户重新输入
            echo "Input error,please yes or no." >2
            continue
            ;;
        esac
    done
fi
```

这样初始化系统网络的脚本就编写完成了。

19.6.2　更改 SSH 服务的端口

在新操作系统中，如果使用不修改 SSH 服务的端口号（默认端口号为 22），可能会给攻击者制造机会。因此安装完操作系统后，建议立即修改 SSH 服务的端口号。当然最好的办法是让脚本来完成这项工作。修改 SSH 服务端口时，先使用 sed 命令修改 SSH 服务的配置文件，然后重启 sshd 服务，并修改防火墙规则：

```
#修改 SSH 服务端口
#modify SSH port
#输出提示信息并备份原始配置文件
echo ""
echo "backup configure file."
TMP_FILE=$BACKUP_CONFIG_DIR/sshd_config.'date +"%Y-%m-%d.%H-%M"'
```

```
cp /etc/ssh/sshd_config $TMP_FILE
#输出提示并将配置文件中的端口配置一行内容进行修改
#修改后的端口为 12345
echo "Modify sshd use to port:12345"
sed 's/\#Port 22/Port 12345/' $TMP_FILE >/etc/ssh/sshd_config
#输出提示信息并重新启动 sshd 服务
echo "Restart sshd service,please wait..."
/etc/init.d/sshd restart &>/dev/null

#修改系统防火墙规则并保存
echo "Modify system firewall rules,please wait..."
/etc/init.d/iptables restart &>/dev/null
iptables -t filter -A INPUT -p tcp --dport 12345 -j ACCEPT
iptables -t filter -A OUTPUT -p tcp --sport 12345 -j ACCEPT
/etc/init.d/iptables save &>/dev/null
echo "OK!"
```

上面的脚本中，先使用 cp 命令备份 SSH 服务的配置文件，然后使用 sed 编辑器修改配置文件内容，并重启 sshd 服务让新配置文件生效。最后还需要添加防火墙规则，否则管理员将无法使用 SSH 服务。

19.6.3　设置 SELinux 状态

SELinux 是一个能增强 Linux 安全性的子系统，但开启 SELinux 可能会致使某些服务发生错误，因此许多管理员都会考虑关闭 SELinux。关闭 SELinux 时，先使用 sed 命令修改配置文件，然后使用 setenforce 命令关闭 SELinux。

在初始化系统脚本中实现这个功能：

```
#关闭 SELinux
#close SELinux
#输出提示并备份原始配置文件
echo ""
echo "Close selinux..."
TMP_FILE=$BACKUP_CONFIG_DIR/selinux.'date +"%Y-%m-%d.%H-%M"'
cp /etc/sysconfig/selinux $TMP_FILE
#修改配置文件并使用命令 setenforce 关闭 SELinux
sed 's/SELINUX=enforcing/SELINUX=disabled/' $TMP_FILE >/etc/sysconfig/
selinux
setenforce 0
echo "OK!"
```

19.6.4　关闭不必要的系统服务

新安装的操作系统中，通常会存在许多不必要的系统服务。运行这些不必要的系统服务会造成系统资源浪费，并且某些不安全的系统服务还会危及系统安全。因此有必要在初始化系统时关闭不必要的服务：

```
#关闭系统中不必要的服务
```

```
#close unused service
echo ""
echo "Close unused service"
#将需要关闭的服务保存至变量中
CLOSE SERVICE="bluetooth cups ip6tables isdn atd acpid hidd hplip"
#使用 for 循环处理变量中保存的服务名
for SERVICE in $CLOSE SERVICE
  do
    #使用 chkconfig 命令设置服务的启动状态为关闭
    chkconfig $SERVICE off
    echo "Close service $SERVICE,please wait..."
    #使用 service 命令关闭服务
    service $SERVICE stop &>/dev/null
    echo "OK!"
done

echo ""
echo "End"

#清除使用过的变量和函数
unset BACKUP CONFIG DIR DIR NET DIR INTER IP ADDR NET MASK NET GW
unset NET GW TRUE DNS PRI DNS SEC TMP FILE ANS
unset input interface input ip addr input gw test conn save inter dhcp
unset input gw error save inter ip input dns save dns
exit 0
```

至此初始化系统脚本就已完成了。读者可以通过修改和完善这个脚本的方法学习脚本编程。修改和完善的方法有：使用 dig 命令测试 DNS 服务器地址，在脚本中添加优化系统功能等。

19.7　服务监控和主备切换脚本

如果管理员管理的服务器一刻也不能中止，最好为服务器设置主备切换。在主备切换中，需要设置两台服务器，一台是正在提供服务的主要服务器，另一台是可用于提供服务的备用服务器。当主要服务器出现问题后，次要服务器会立即启动服务，接替出错的主要服务器继续提供服务。这就是服务器的主备切换。

目前在 Web 服务器中能实现这种需求的技术有许多，例如 HAProxy、F5、HA 等，其工作原理也有所差异，读者可参考相关文档。本节中将自己动手编写用于 Web 服务器的服务监控和主备切换的简单脚本。

19.7.1　测试主服务器状态

只有当主服务器出现问题才能切换备用服务器状态，因此首先需要测试主服务器的状态。测试服务器状态使用 wget 命令，以下载的方式测试主服务器的 Web 服务是否能正常工作。

在本例中测试主服务器状态的函数如下：

```
#!/bin/bash

#This script is used to monitor the status of the master server.

#定义主服务器的 IP 地址变量、日志文件变量
#master ip address
MASTER_IP="192.168.118.100"
#log file
FILE_LOG=/root/master_status.log

#用于测试主服务器状态的变量
#function test_master
function test_master()
{
   #使用 wget 命令检测主服务器的 Web 服务是否可用
   wget $MASKTER_IP &>/dev/null
   return $?
}
```

在上面的函数中，使用 wget 工具测试主服务器的 Web 功能，并根据 wget 工具的返回值返回函数。

注意：脚本中的主服务器 IP 地址并非真实地址，仅供示例参考。

19.7.2 切换备用服务器状态

当脚本检测到主要服务器发生故障不能提供服务时，就需要切换备用服务器的状态。切换备用服务器状态时，先使用 ifconfig 命令为接口添加子接口，然后再重启 httpd 服务。

（1）在本例中切换备用服务器状态函数的内容如下：

```
#定义启动备用服务的函数
#function start savle server
function start savle()
{
   #使用 ifconfig 命令为 eth0 接口添加子接口
   /sbin/ifconfig eth0:1 $MASTER IP/24 up
   #启动 httpd 服务
   /etc/init.d/httpd restart &>/dev/null
   return $?
}
```

（2）完成测试函数和切换备用服务器状态函数后，就需要编写脚本的主体部分。在脚本的主体部分中，使用 while 语句实现循环监控。在循环监控中，先使用 test_master 函数测试主服务器状态，如果测试失败，就立即调用切换备用服务器状态函数启动备用服务器。

脚本主体部分内容如下：

```
#使用循环监控主要服务器
```

```
while true
  do
    #调用测试函数测试主服务器状态
    test master
    #判断主服务器状态，如果主服务器可以提供服务，就停止执行 10 秒
    if [ $? = 0 ]
      then
        sleep 10
    else
        #如果主服务器无法提供服务就写入日志，并调用 start_savle 函数启动备用服务
        echo 'date' >$FILE LOG
        echo "Master server fails." >$FILE LOG
        echo "time:"'date' >/tmp/master.tmp
        echo "Master server fails." >>/tmp/master.tmp
        start savle
        #如果备用服务启动成功，就写入日志并发送邮件给管理员
        if [ $? = 0 ]
          then
            echo "Savle server start successfully." >$FILE LOG
            echo "End" >$FILE LOG
            echo "Savle server start successfully." >>/tmp/master.tmp
            mail -s "Master server fails" root </tmp/master.tmp
            rm -rf /tmp/master.tmp
            exit 0
        else
            #如果备用服务启动失败就写入日志，并发送邮件给管理员
            echo "Savle server fails to start">$FILE LOG
            echo "End" >$FILE LOG
            echo "Savle server fails to start" >>/tmp/master.tmp
            mail -s "Master server fails,savle server fails to start" root
            </tmp/master.tmp
            rm -rf /tmp/master.tmp
            mail -s
        fi
   fi
done
```

上面只是一个非常简单的示例，读者可以通过修改、完善示例脚本的功能学习脚本编程。修改和完善的方向有：添加网络测试、添加主服务器恢复测试等。

19.8　小　　结

- 19.1 节主要介绍了 Linux 系统中的运行级别脚本，以及如何编写运行级别脚本等内容。学习运行级别脚本后，用户将自己编写的脚本添加到系统中，非常便于管理员监控和管理系统。
- 19.2 节讲解了编写脚本的原则、编程过程中的良好习惯和脚本安全性等内容，学习这些内容有利于初学者编写出高质量的脚本。
- 19.3 节介绍了如何调试脚本，调试脚本的原则、注意事项和工具等。

- 19.4 节利用编写备份脚本的过程，讲解了编写脚本的思路和过程。
- 19.5 节介绍了快速配置防火墙脚本，介绍了顺序执行类脚本的应用。
- 19.6 节引用快速初始化系统脚本的编写过程，讲解了 Shell 脚本中不同的测试方法。
- 19.7 节引用监控、切换主备脚本的编写过程，介绍了与服务相关的脚本编写过程。

在本章中，引用了许多脚本示例介绍脚本在系统管理过程中的应用。在每个示例结尾都提出了脚本修改和完善意见，读者可以参照自己的需求修改脚本，在示例中学习如何编写脚本。